**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Российский экономический университет имени Г. В. Плеханова»**

Высшая школа кибертехнологий, математики и статистики

Кафедра математических методов в экономике

Выпускная квалификационная работа

по программе профессиональной переподготовки

«Основы Data Science на языке Python»

на тему «Создание MVP по оценке сходства с лицами знаменитостей»

Выполнил:

Коротков Фёдор Олегович

Преподаватель:

профессор кафедры математических методов в экономике,

д.э.н. Моисеев Никита Александрович

Москва

2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc129729942)

[1 Анализ предметной области и постановка задачи 4](#_Toc129729943)

[1.1 Постановка задачи 4](#_Toc129729944)

[1.2 Доступные ресурсы 4](#_Toc129729945)

[1.3 Риски 5](#_Toc129729946)

[1.4 Ограничения 5](#_Toc129729947)

[1.5 Критерии успешности 5](#_Toc129729948)

[2 Создание структуры MVP, ETL-процесса 7](#_Toc129729949)

[2.1 Создание структуры MVP 7](#2.1_Создание_структуры_MVP)

[2.2 Организация ETL-процессов 7](#_Toc129729952)

[3 Моделирование 17](#_Toc129729957)

[3.1 Тривиальная модель 17](#_Toc129729959)

[3.2 Модель GARCH 18](#_Toc129729960)

[3.3 Модель LSTM 19](#_Toc129729961)

[3.4 Модель GRU 22](#_Toc129729962)

[3.5 Оценка результатов 25](#_Toc129729963)

[ЗАКЛЮЧЕНИЕ 28](#_Toc129729964)

[Приложение А (обязательное). Исходный код программы 29](#_Toc129729965)

# ВВЕДЕНИЕ

В настоящее время одним из наиболее востребованных направлений работы с ИИ являются нейронные сети и в частности задачи практического применения CV (computer vision).

В качестве основной задачи данной выпускной квалификационной работы определена задача определения процента схожести черт тестируемого лица с лицами знаменитостей мирового уровня с использованием методов глубокого машинного обучения — библиотек CV.

Основная цель работы – показать полный процесс создания и деплоя в публичный доступ в сети интернет MVP (minimum viable product) по определению процента схожести тестируемого лица с лицами из базы данных, на которой обучена наша модель.

В качестве дополнительных целей, хотелось бы выделить постановку приоритета в реализации MVP на поддерживаемость и масштабируемость проекта, а так же достаточно легкую возможность перепрофилировать его в более узкоспециализированные решения, например создание базы данных лиц сотрудников компании и использовании в системе фиксации рабочего времени сотрудника или организации разных уровней допуска в соответсвующие помещения организации.

Исходный код программы приведён в приложении А.

# 1 Анализ предметной области и постановка задачи

# Постановка задачи

К задачам создания MVP относятся:

- создание общей структуры проекта;

- организация ETL-процесса (extract, transform, load), то есть процесса извлечения (сбора) данных их преобразования и загрузки в базу данных проекта;

- построение модели по определению процента схожести тестируемого лица;

- оценка метрик полученной модели, локальное тестирование модели;

- деплой модели в публичный доступ в сеть интернет;

- тестирование и интерпретация полученной MVP;

- выгрузка как open-source project в публичный репозиторий GitHub.

# Доступные ресурсы

Для выполнения задачи доступны следующие ресурсы:

- язык программирования Python3.8 со следующими библиотеками (так же указаны в файле requirements.txt для более быстрой и удобной установки): pandas==2.0.0, numpy, matplotlib, Google-Images-Search, bing-image-downloader, Pillow, tensorflow, face\_recognition, scikit-learn, Flask;

- парсинг фотографий лиц знаменитостей из интернета;

- знания, полученные в ходе обучения, а также статьи и публикации, youtube.com уроки относящиеся к теме CV в части распознаванию лиц;

- среда разработки PyCharm;

- условно бесплатный хостинг ресурс для пайтон скриптов от проекта Anaconda – pythonanywhere.com

- облачный репозиторий для разработчиков Github.

# Риски

В ходе выполнения данной работы существуют определенные риски, которые могут повлиять на точность результатов. В основном, большая часть рисков так или иначе обусловлена применением скриптов для автоматизации поиска фотографий для наполнения базы данных модели. Выделим основные «узкие места» применения скриптов и возможные последствия.

Один из главных рисков — это риск моделирования на некорректном наборе данных, то есть модель будет обучаться на фотографиях, которые принадлежат не тем лицам, которые подразумевались или на фотографиях отсутствуют лица вообще.

Существует риск снижения качества классификации конкретных знаменитостей из-за малого количества собранных изображений в виду технических причин: блокировка сайтом русского айпи, с которого ведется парсинг фото, ошибка загрузки фотографий из-за технических сбоев в соединении, антипарсинг системы типа cloudflare и др.

Также хотелось бы выделить в качестве «бутылочного горлышка» проблему квоты дискового пространства на условно-бесплатном хостинге pythonanywhere.com, которое наложило существенные ограничения на логику работы приложения — пришлось заменить часть кода, который изначально предполагал выбор случайной картинки из имеющейся базы данных знаменитостей и показа в конечном итоге для визуального сравнения результатов работы модели, на поисковый запрос в интернет и выдачу случайной фотографии лица знаменитости непосредственно в ходе рабочего процесса, что сказалось на скорости работы сервиса.

# Ограничения

В ходе выполнения работы существует ряд ограничений, который может повлиять на выполнение работы:

- используются сторонние нейронные сети для решения задачи распознавания лиц на фотографиях и получения их эмбеддингов, что непосредственно влияет на качество конечной модели;

- ограниченность в вычислительных, временных и hardware ресурсах, что оказывается влияние на время осуществления ETL-процессов, обучение модели и, как следствие, точности прогнозов.

# Критерии успешности

Для оценки качества модели, необходимо сравнить реальные и предсказанные значения и рассчитать следующие метрики:

- Precision — точность;

- Recall — полнота;

- f\_1 score — f\_1 мера

- MSE (Mean Squared Error) – среднеквадратическая ошибка;

- MAE (Mean Absolute Error) – средняя абсолютная ошибка;

- MAPE (Mean Absolute Percentage Error) — средний абсолютный процент ошибки.

Для оценки работоспособности MVP — тест модели развернутой в публичном доступе в сети интернет.

# 2 Создание структуры MVP, ETL-процесса



# Создание структуры MVP

Общая структура проекта представлена на рисунке 1.

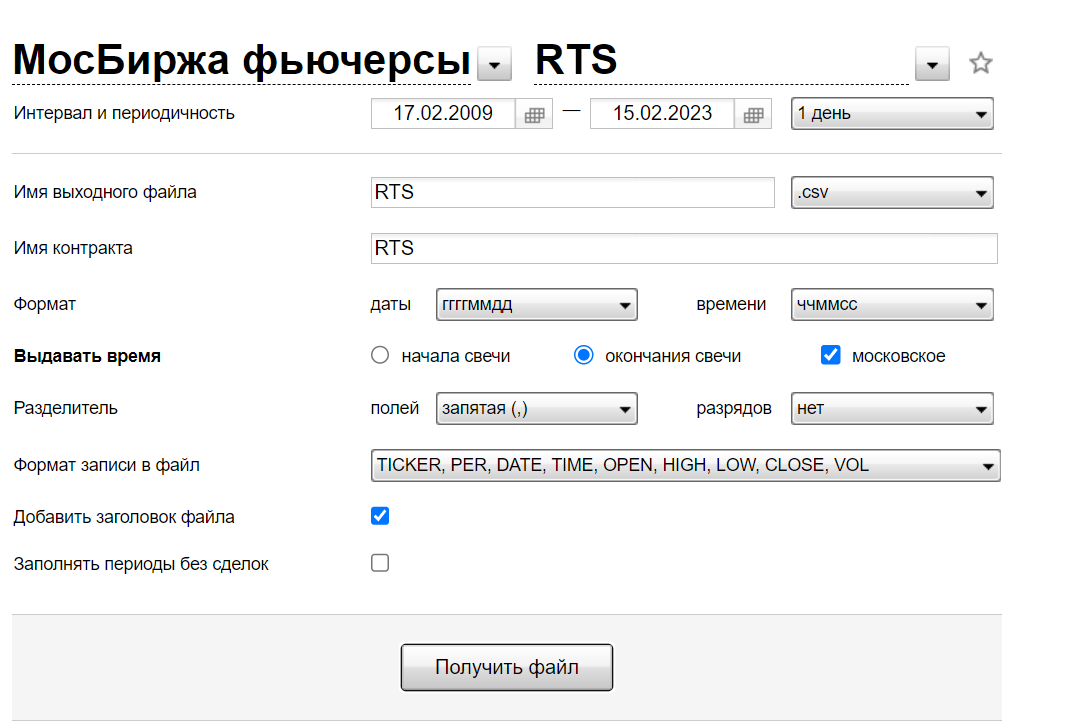


Рис. 1 – Общая структура проекта

Ключевыми блоками являются блоки ETL-процесса формирования базы данных изображений для последующего обучения модели, а так же блок создания модели и оценки ее качества. Остальные блоки не столь важны и могут легко варьироваться в зависимости от конкретных конечных целей профилирования MVP.

# Организация ETL-процессов

Принципиальная схема сбора, обработки и выгрузки в локальную базу данных представлена на рисунке 2.

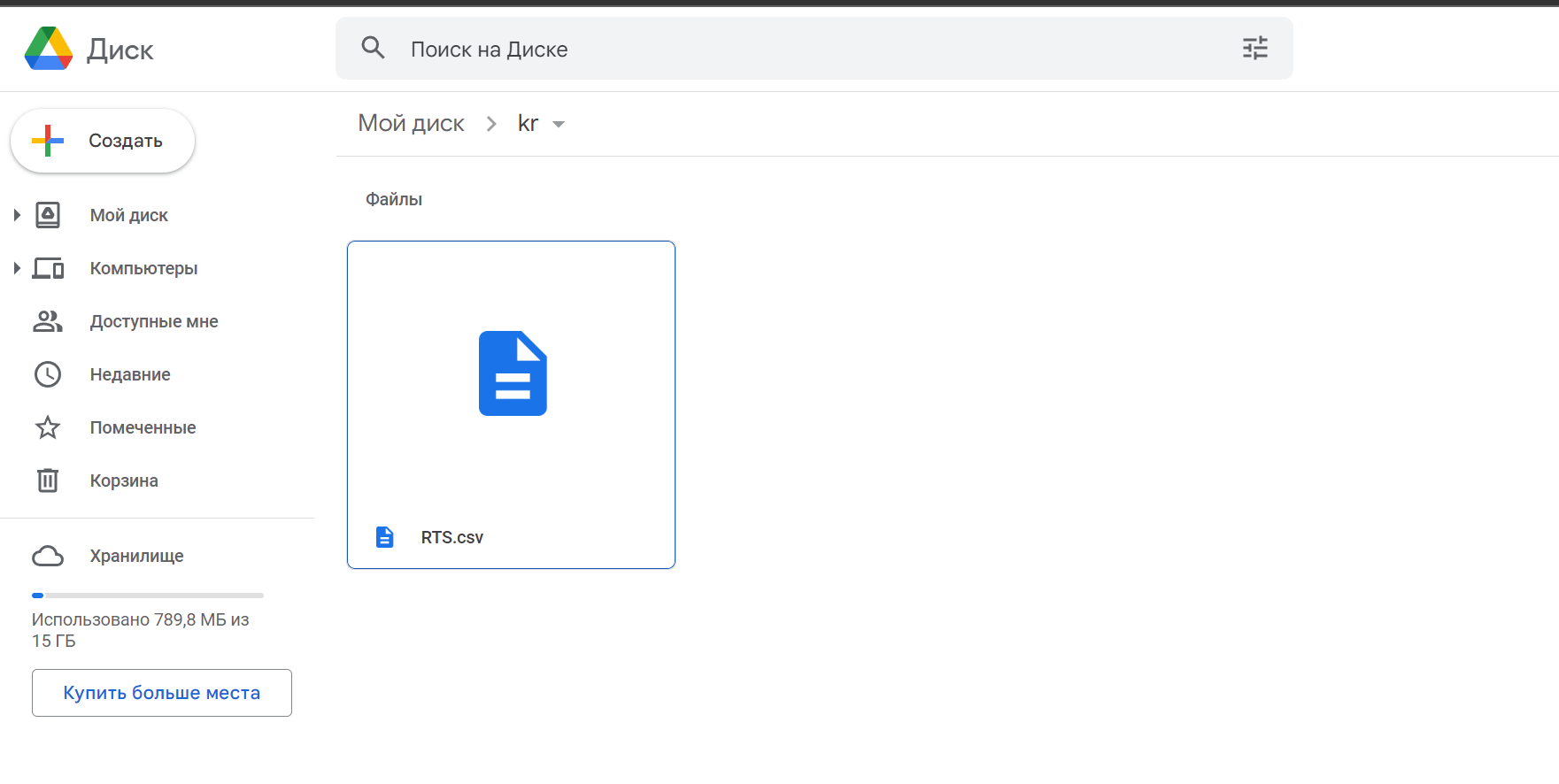


Рис. 2 – Схема организации ETL-процесса

Этап парсинга изображений был автоматизирован с помощью созданного скрипта и библиотеки bing\_image\_downloader.

На этапе преобразования данных использовались библиотеки:

- PIL (форк от Pillow) — преимущественно для открытия спарсенных изображений и перевода в RGB — цветовой канал.

- tensorflow — для нормализации размера изображений (по наибольшей стороне к заданному размеру. В нашем случае использовался размер в 256 пикселей). Так же данная библиотека использовалась для проведения процесса аугментации нашего набора данных — в нашем проекте использовался метод вертикального зеркалирования нормализованного изображения.

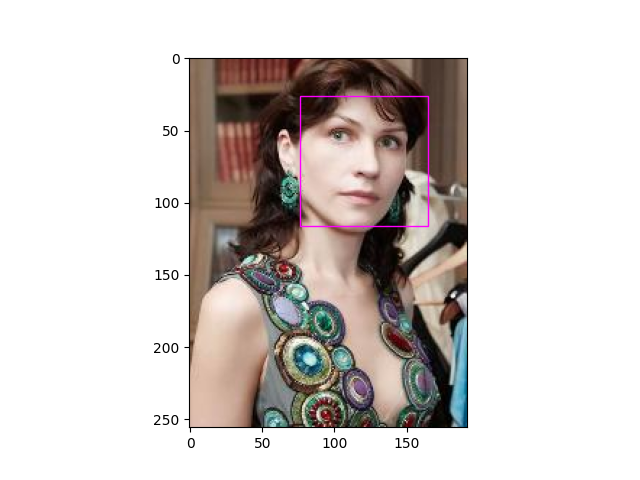
Благодаря использованию аугментации мы увеличили размер нашего набора данных вдвое — до 2770 изображений.

Для создания конечного набора данных (pandas data set), на котором предполагается последующее обучение модели был написан отдельный скрипт. Данное решение обусловлено желанием добиться независимости и гибкости в процессах редактирования базы данных изображений и будущими дообучениями моделей.

Для создания финального дата сета была использована библиотека для распознавания лиц — face\_recognition.

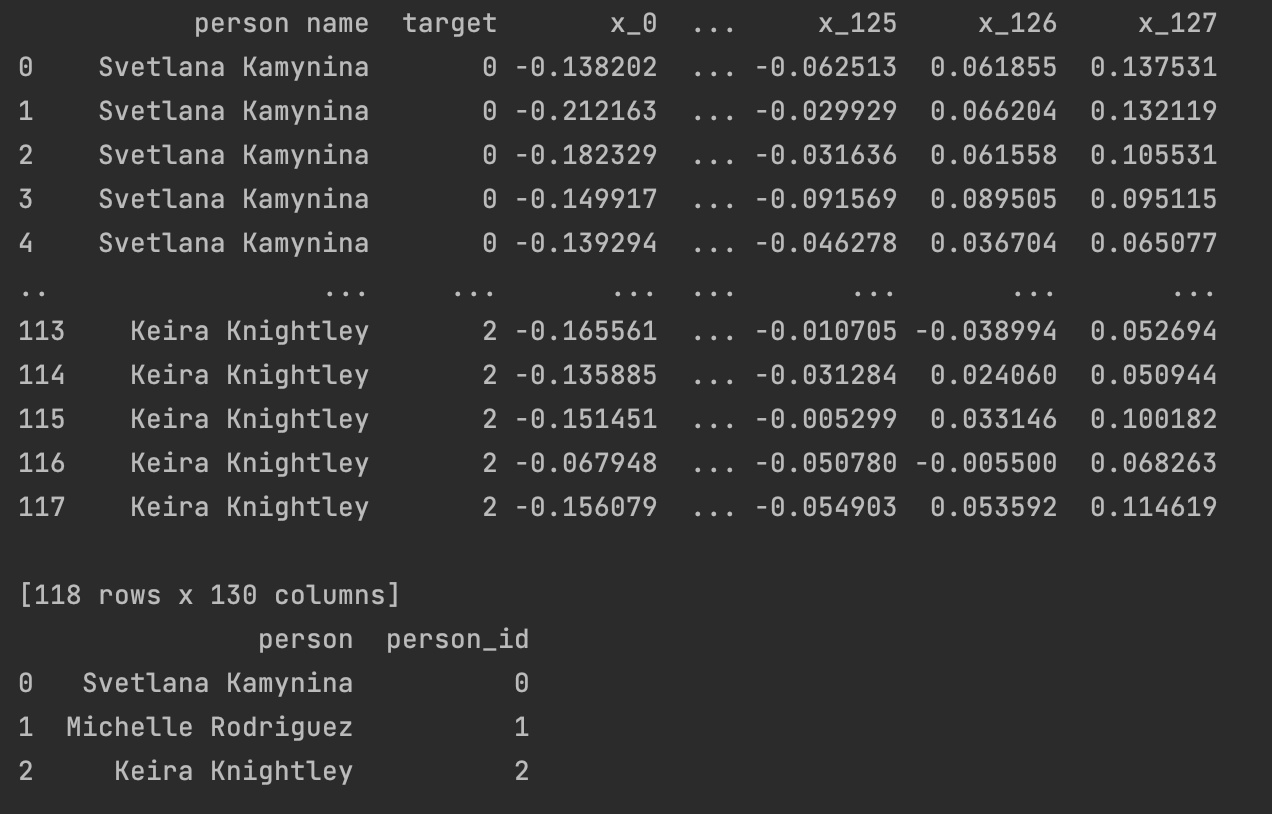
При использовании данной библиотеки с помощью ее нейронных сетей на каждом изображении отыскивалось лицо или лица и получались координаты рамок вокруг ключевых фич, характеризующие найденные лица. В случае, если находилось более одного лица (истинно или ложно) такое изображение удалялось из последующей обработки и включения в финальный дата сет.

На рисунке 3 представлен пример работы данной библиотеки и локализации найденного лица.

Рис. 3 – Пример локализации лица с помощью face\_recognition

Далее, необходимо произвести кодирование полученной области в многомерный вектор, который будет описывать фичи полученной области. Это так же было сделано с помощью face\_recognition и встроенного метода encoding. Полученный вектор называется эмбеддингом (embedding) данной локализации изображения и имеет 128-мерный размер. Таким образом, наш дата сет будет содержать 128 фич для описания каждой целевой переменной — person\_id. Кроме того, имеет смысл добавить в финальный дата сет категориальную переменную — имя и фамилию знаменитости.

Таким образом, наш финальный дата сет имеет 130 столбцов: имя и фамилию знаменитости, уникальный идентификатор знаменитости и 128 фичей, взятых из разложенного по элементам эмбеддинга. Так же в процессе создания финального дата сета был создан матчинговый дата сет, в котором содержатся только имя и фамилия знаменитости и его уникальный номер. Это нужно для того, чтобы более легко выводить во фронтенд результат матчинга. Примеры дата сетов представлены на рисунке 4.

Рис. 4 – Пример структуры финального дата сета (вверху) и матчингового (внизу)

Завершающим этапом было сохранение двух дата сетов с помощью встроенного python-модуля pickle.

# 3 Моделирование

В качестве основной модели была выбрана модель логистической регрессии, поскольку при таком количестве фич и размере финального дата сета ее функционала более чем достаточно. В дальнейшем эта модель может быть использована в качестве бейзлайна для сравнения с другими моделями либо тюнингом текущей.

Создание модели было вынесено в отдельный скрипт.

Этапы моделирования следующие (рисунок 5):

- загрузка с помощью pickle финального и матчингового дата сетов;

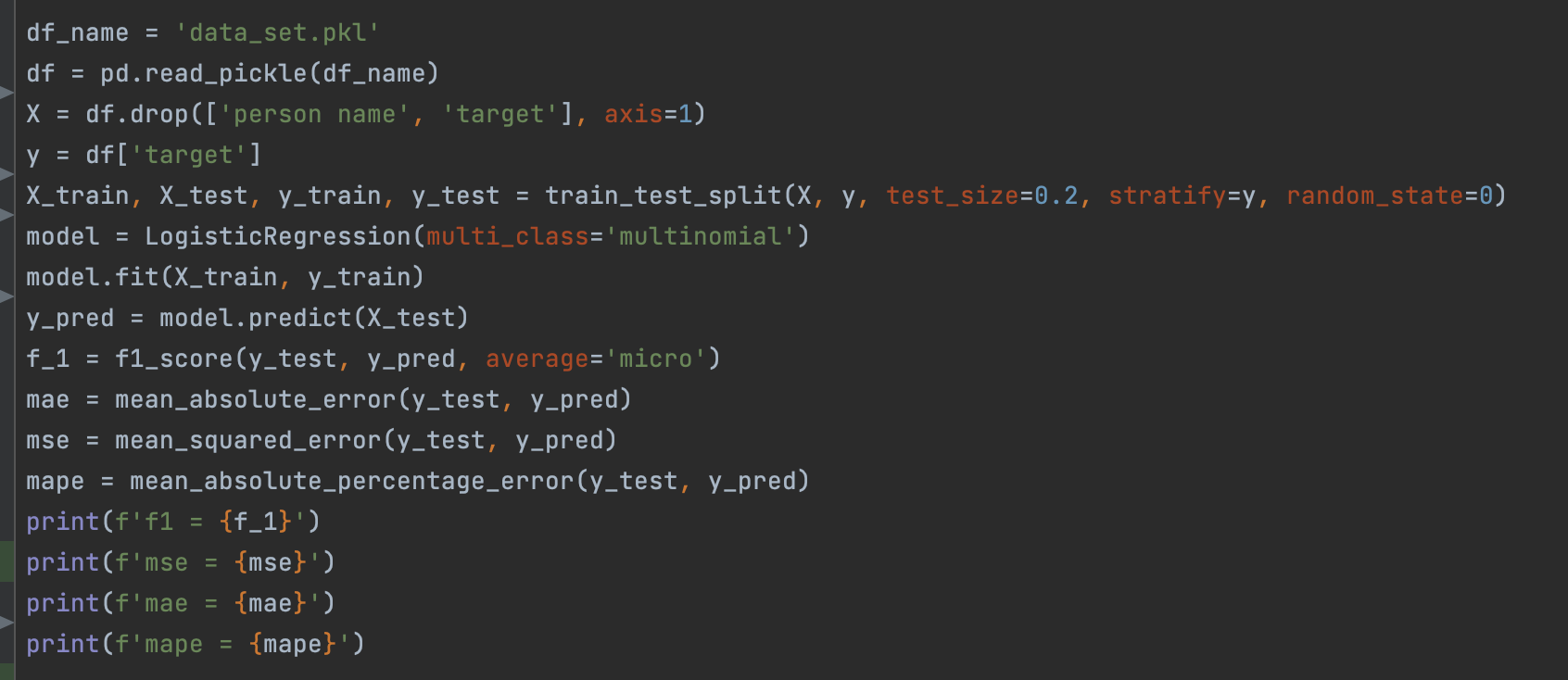
- формирование дата сета фичей из финального путем выброса двух столбцов — имя и фамилия знаменитости и его id;

- формирования дата сета с таргетами — это колонка с id;

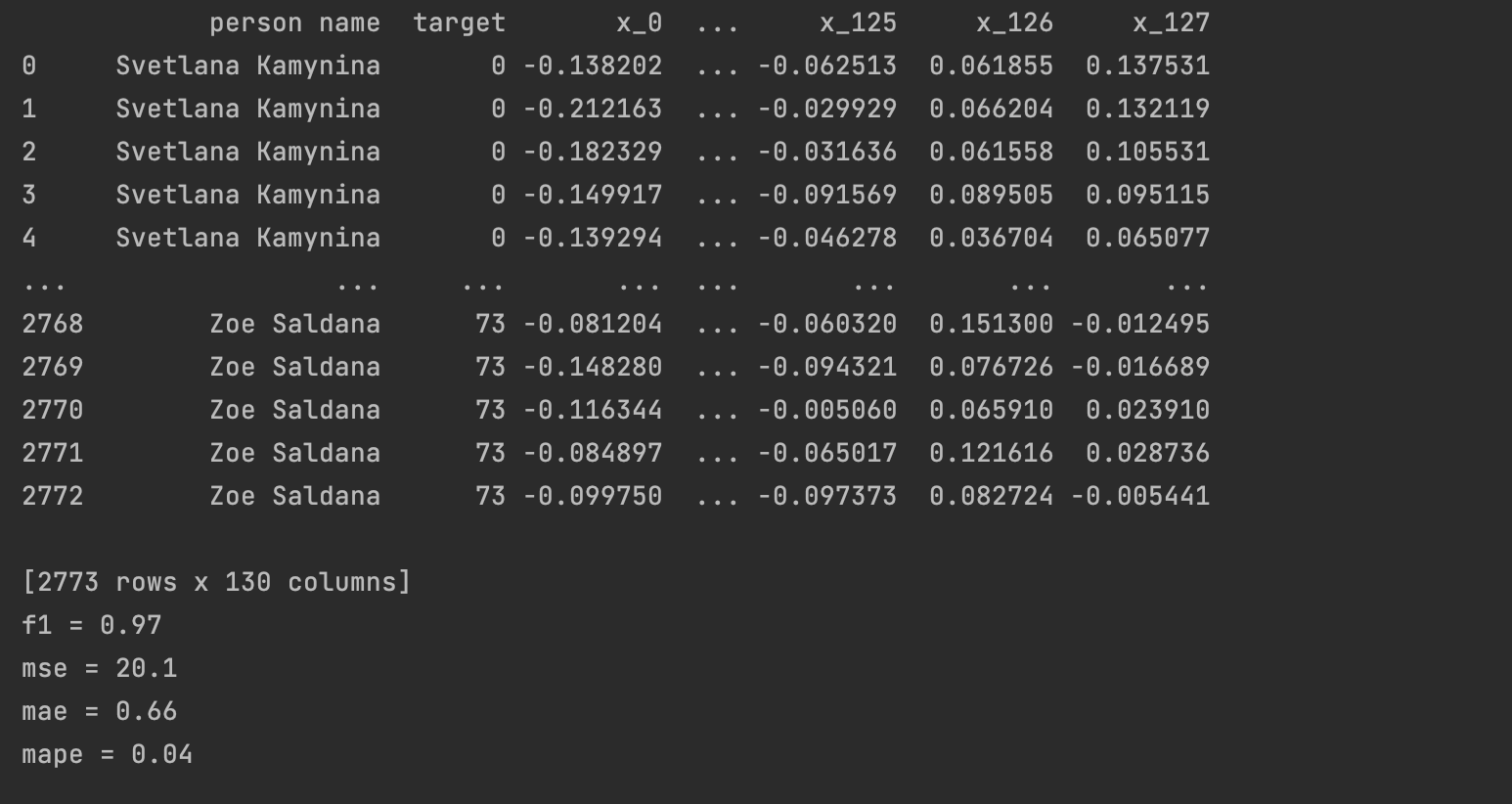
- разбиение всех дата сетов на тренировочные и тестовые выборки в соотношении 80% на 20% и стратификации по таргету — чтобы мы были уверены, что выборки будут более-менее одинаково наполненные;

- обучение модели в многоклассовом режиме;

- расчет (с усреднением по методу micro) и оценка метрик получившейся модели.

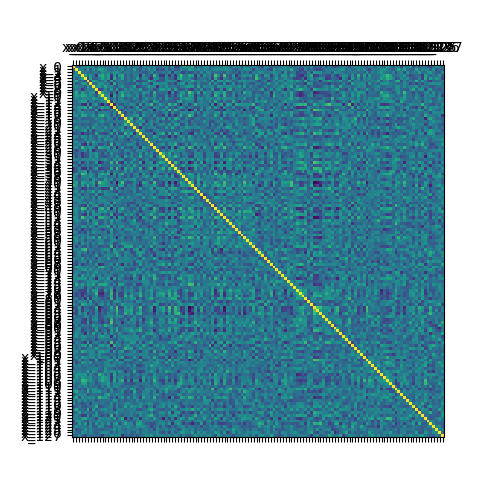
Рис. 5 — Основные этапы построения модели логистической регрессии

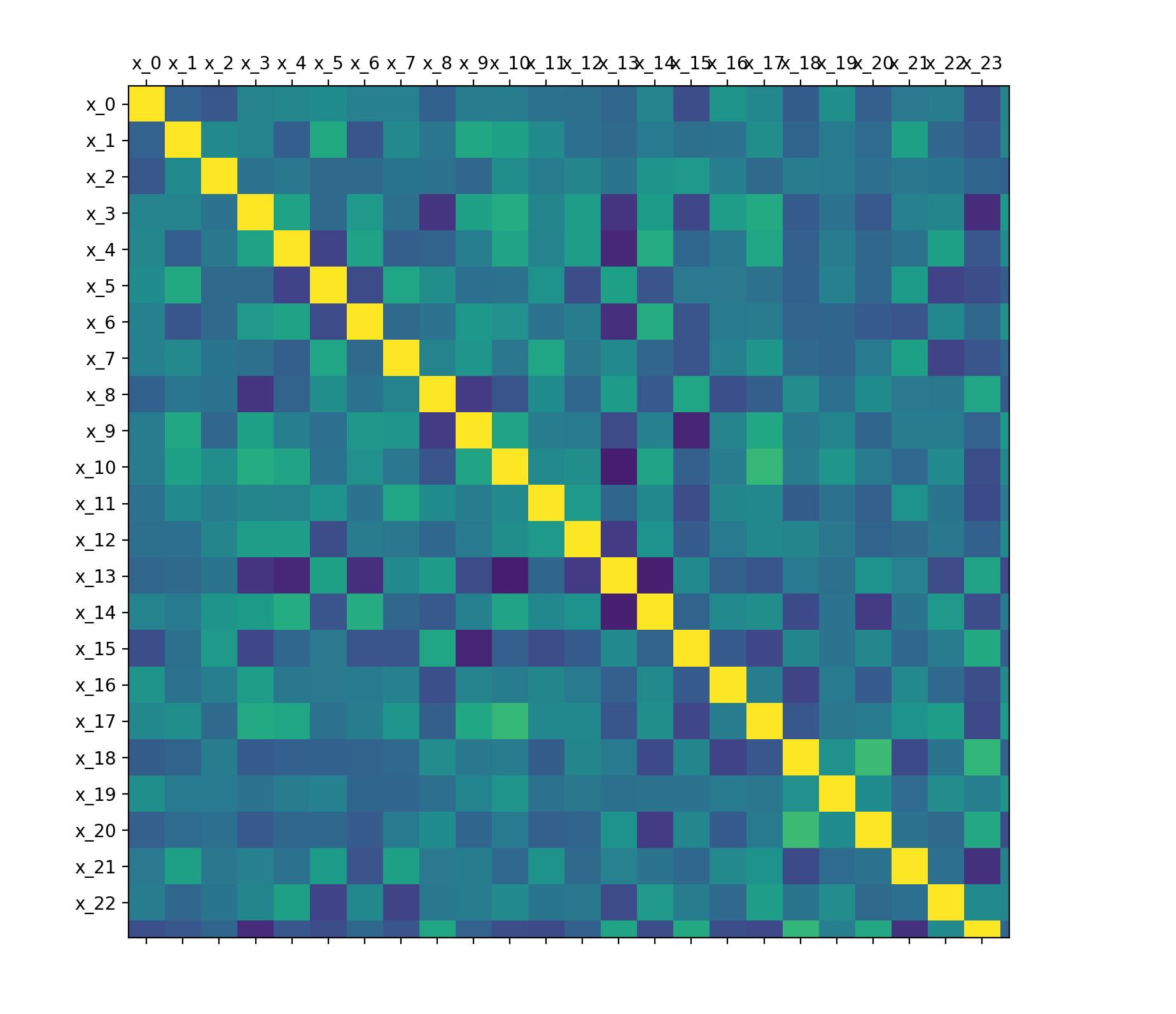
Финальный дата сет и метрики модели представлены на рисунке 6.

Рис. 6 — Фрагмент финального дата сета и посчитанные метрики модели

F1 — метрика модели достаточно высокая, что обусловлено большим количеством фичей, при небольшом количестве таргетов (128 на 73 соответственно). Таким образом, модель достаточно хорошо оптимизирована.

Корреляционная матрица и укрупненный фрагмент представлены на рисунке 7 и 8. Видно, что иногда некоторые фичи имеют высокую мультиколлинеарность, что обусловлено спецификой фич — кодированием цвета.

Рис. 7 — корреляционная матрица (полная)

Рис. 8 - укрупненный фрагмент корреляционной матрицы

Таким образом, в качестве бейзлайн-модели мы можем использовать данную модель в целях реализации MVP.

# Модель GARCH

Модель GARCH (Обобщенная авторегрессионная модель гетероскедастичности) – модель для анализа временных рядов, которая позволяет прогнозировать волатильность.

На рисунке 19 представлен код построения модели GARCH

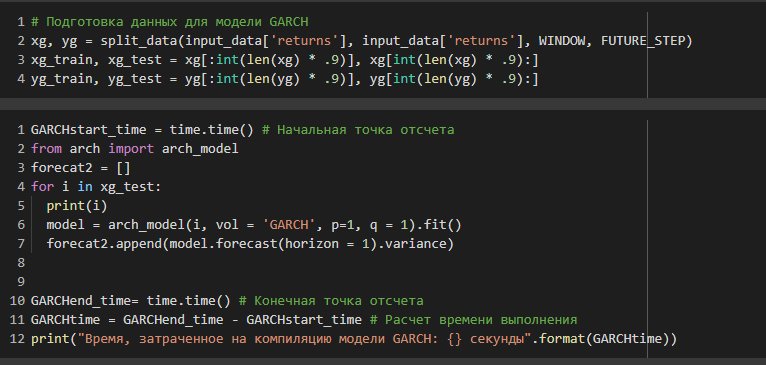


Рисунок 19 – Код построения модели GARCH

Данная модель не нуждается в отдельном обучение, так как сразу обучается и делает предсказание на заданное количество шагов вперед. Для нашей задачи будут использовать тестовые данные, для последующего сравнения результатов с остальными моделями.

На рисунке 20 представлен результат прогноза модели GARCH.

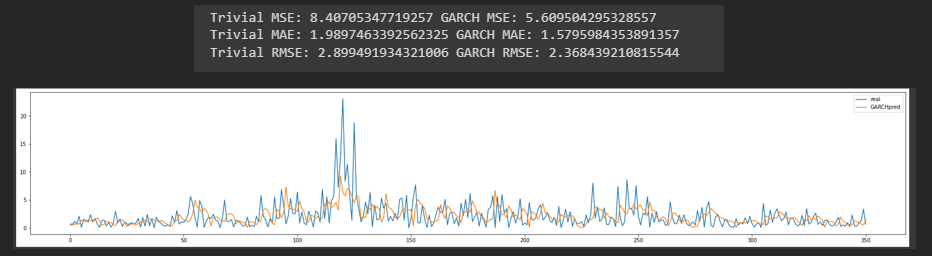


Рисунок 20 – Результат прогноза модели GARCH

# Модель LSTM

3.3.1 Построение модели LSTM

Модели LSTM и GRU – это модели рекуррентных нейронных сетей. Для сравнения моделей, построим идентичные модели LSTM и GRU.

На рисунке 21 представлен код модели LSTM.

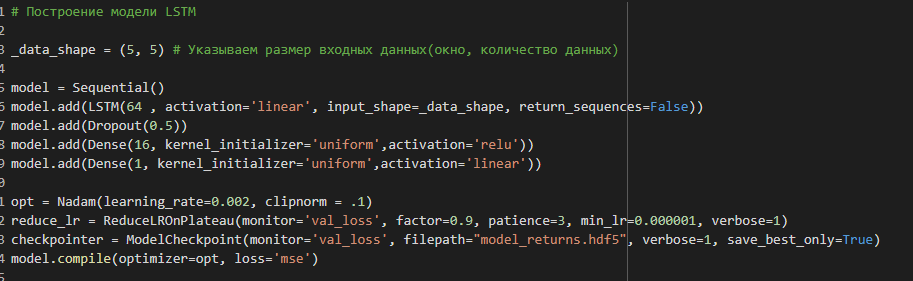


Рисунок 21 – Код модели LSTM

3.3.2 Описание архитектуры модели:

- входные данные имеют размерность (5, 5) — это означает, что входные данные состоят из пяти последовательностей, каждая из которых содержит пять значений;

- cлой LSTM (Long Short-Term Memory) с 64 нейронами используется для анализа последовательности данных и выделения важных признаков;

- Dropout ("регуляризация") с коэффициентом отсечения 0.5 применяется для предотвращения переобучения модели;

- cлой Dense с 16 нейронами, используется для преобразования выходных данных LSTM в новые признаки;

- еще один слой Dense с одним нейроном, используется для получения финального выходного значения (прогноза);

- оптимизатор Nadam используется для обучения модели. Он сочетает в себе методы Nesterov accelerated gradient descent и Adam;

- функция потерь, используемая в модели - среднеквадратичная ошибка (MSE);

- ReduceLROnPlateau — это колбэк (callback), который используется для уменьшения скорости обучения (learning rate), когда происходит стагнация в процессе обучения;

- ModelCheckpoint — это колбэк, который используется для сохранения модели с наименьшим значением функции потерь (MSE) на валидационных данных.

3.3.3 Обучение модели и прогноз

Обучать модель будем на тренировочных данных.

На рисунке 22 представлен код для обучения модели LSTM

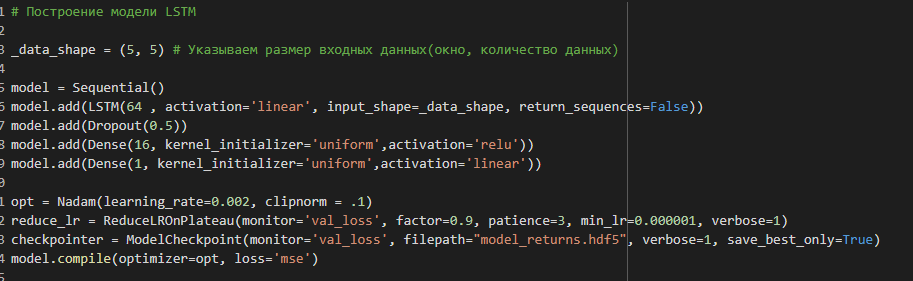


Рисунок 22 – Код для обучегтя модели LSTM

Этот код обучает модель LSTM с использованием библиотеки Keras. Рассмотрим его подробнее:

- LSTMstart\_time = time.time() и LSTMend\_time = time.time() - это начальная и конечная точки отсчета времени, используемые для измерения времени, затраченного на обучение модели;

- history = model.fit(...) — это метод обучения модели. Он использует обучающий набор данных x\_train и y\_train, разбивая его на пакеты размера batch\_size и обучая модель на epochs эпохах. Также он использует валидационный набор данных, указанный в validation\_split, и сохраняет результаты обучения в history для последующего анализа:

- callbacks = [reduce\_lr, checkpointer] — это список обратных вызовов, используемых для отслеживания обучения модели. reduce\_lr — это обратный вызов, который будет уменьшать скорость обучения, если точность на валидационном наборе данных не улучшается в течение нескольких эпох.

- checkpointer — это обратный вызов, который сохраняет модель после каждой эпохи обучения;

- verbose=1 - это параметр, который управляет тем, какая информация будет выводиться в процессе обучения модели. verbose=1 означает, что будет выводиться информация о каждой эпохе обучения;

- print("Время, затраченное на компиляцию модели LSTM: {} секунды".format(LSTMtime)) - это строка, которая выводит время, затраченное на обучение модели. Она использует переменную LSTMtime, которая содержит разницу между начальной и конечной точками отсчета времени, чтобы вычислить время, затраченное на обучение модели.

Для предсказания данных по модели создадим переменную LSTM\_predictions и сделаем предсказание на основе x\_test тестовой выборки.

На рисунке 23 представлен результат прогноза модели LSTM.

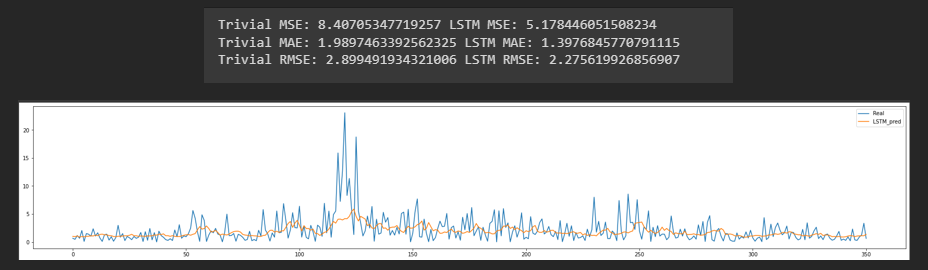


Рисунок 23 – Результат прогноза модели LSTM

# Модель GRU

3.4.1 Построение модели GRU

Модели LSTM и GRU – это модели рекуррентных нейронных сетей. Для сравнения моделей построим идентичные модели LSTM и GRU.

На рисунке 24 представлен код модели GRU.

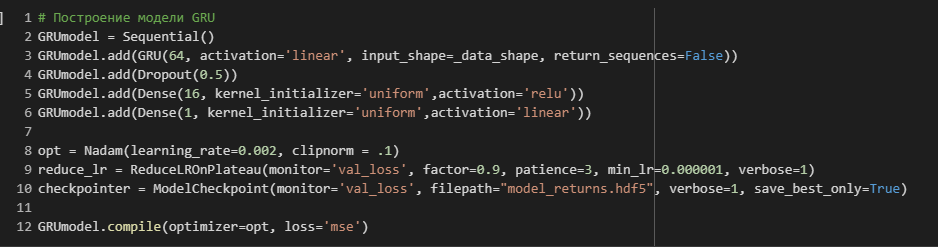


Рисунок 24 – Код модели GRU

3.4.2 Описание архитектуры модели:

- входные данные имеют размерность (5, 5) — это означает, что входные данные состоят из пяти последовательностей, каждая из которых содержит пять значений;

- cлой GRU (Gated Recurrent Unit) с 64 нейронами используется для анализа последовательности данных и выделения важных признаков.;

- Dropout ("регуляризация") с коэффициентом отсечения 0.5 применяется для предотвращения переобучения модели;

- cлой Dense с 16 нейронами, используется для преобразования выходных данных LSTM в новые признаки;

- еще один слой Dense с одним нейроном, используется для получения финального выходного значения (прогноза);

- оптимизатор Nadam используется для обучения модели. Он сочетает в себе методы Nesterov accelerated gradient descent и Adam;

- функция потерь, используемая в модели - среднеквадратичная ошибка (MSE);

- ReduceLROnPlateau — это колбэк (callback), который используется для уменьшения скорости обучения (learning rate), когда происходит стагнация в процессе обучения;

- ModelCheckpoint — это колбэк, который используется для сохранения модели с наименьшим значением функции потерь (MSE) на валидационных данных.

3.4.3 Обучение модели и прогноз

Обучать модель будем на тренировочных данных.

На рисунке 25 представлен код для обучения модели GRU.

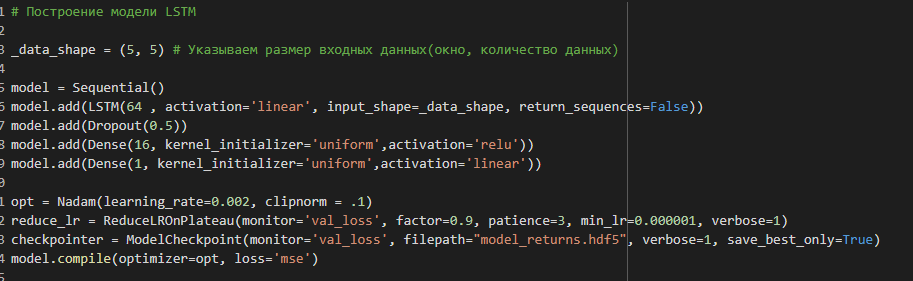


Рисунок 25 – Код для обучения модели GRU

Этот код обучает модель GRU с использованием библиотеки Keras. Рассмотрим его подробнее:

- GRUstart\_time = time.time() и GRUend\_time = time.time() - это начальная и конечная точки отсчета времени, используемые для измерения времени, затраченного на обучение модели;

- history = model.fit(...) — это метод обучения модели. Он использует обучающий набор данных x\_train и y\_train, разбивая его на пакеты размера batch\_size и обучая модель на epochs эпохах. Также он использует валидационный набор данных, указанный в validation\_split, и сохраняет результаты обучения в history для последующего анализа:

- callbacks = [reduce\_lr, checkpointer] — это список обратных вызовов, используемых для отслеживания обучения модели. reduce\_lr — это обратный вызов, который будет уменьшать скорость обучения, если точность на валидационном наборе данных не улучшается в течение нескольких эпох.

-checkpointer — это обратный вызов, который сохраняет модель после каждой эпохи обучения;

- verbose=1 — это параметр, который управляет тем, какая информация будет выводиться в процессе обучения модели. verbose=1 означает, что будет выводиться информация о каждой эпохе обучения;

- print("Время, затраченное на компиляцию модели GRU: {} секунды".format(GRUtime)) - это строка, которая выводит время, затраченное на обучение модели. Она использует переменную GRUtime, которая содержит разницу между начальной и конечной точками отсчета времени, чтобы вычислить время, затраченное на обучение модели.

Для предсказания данных по модели создадим переменную GRU\_predictions и сделаем предсказание на основе x\_test тестовой выборки.

На рисунке 26 представлен результат прогноза модели GRU.

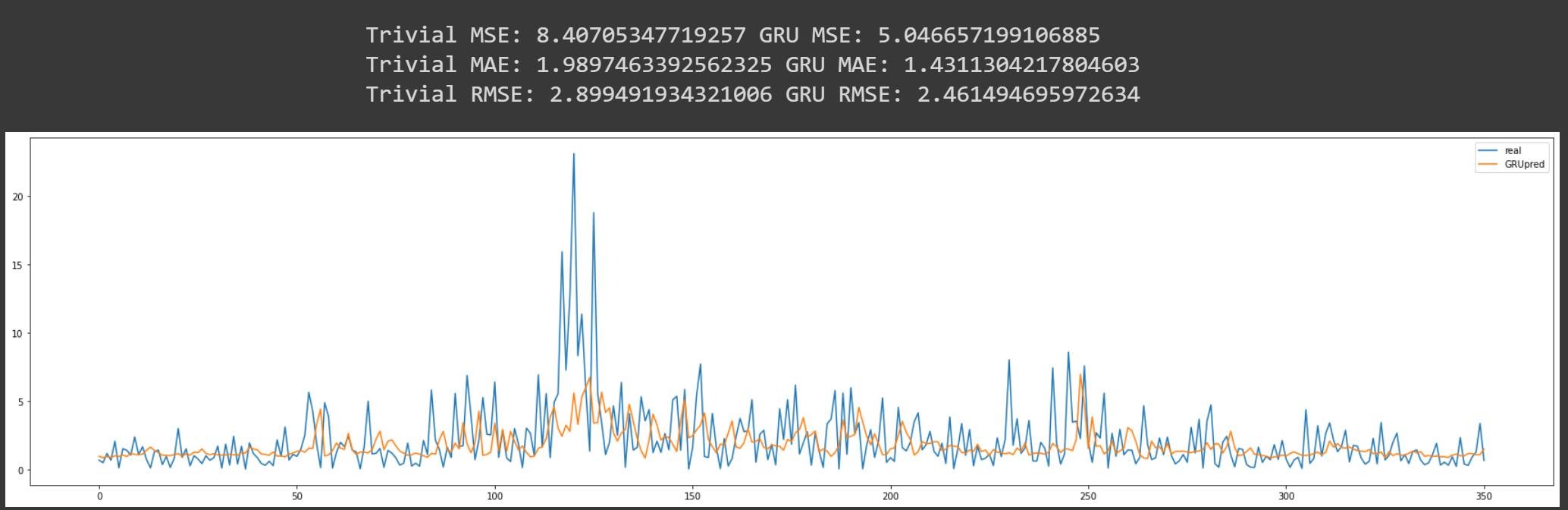


Рисунок 26 – Результат прогноза модели GRU

# Оценка результатов

3.5.1 Оценка результатов моделей

Для оценки результатов создадим датафрейм с реальными значениями и предсказанными.

На рисунке 27 представлен датафрейм с реальными и предсказанными значениями каждой модели.

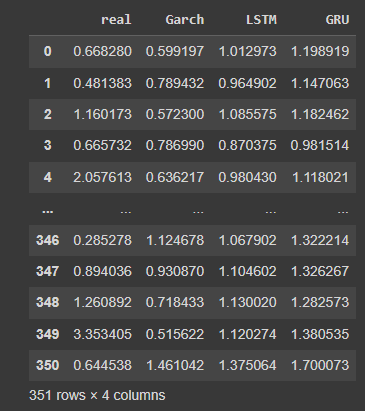


Рисунок 27 – Предсказанные значения модели

Выведем сводных график по данным.

На рисунке 28 представлен сводных график предсказанных и реальных значений волатильности индекса RTS.

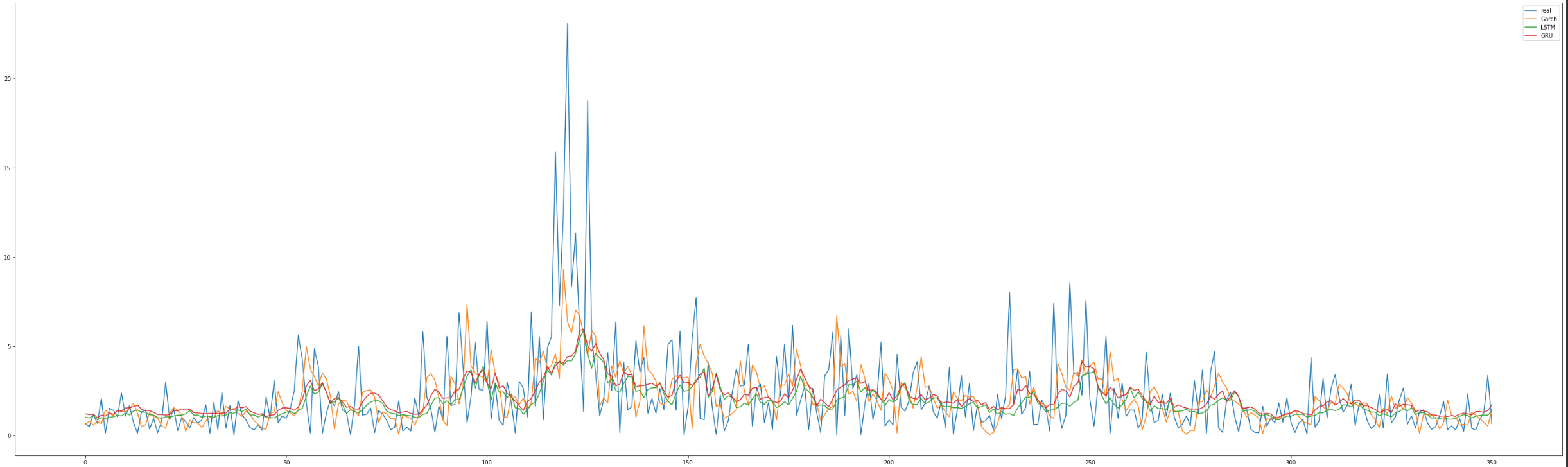


Рисунок 28 – Реальная и предсказанные волатильность RTS

Исходя из графика можно сделать вывод: модели LSTM и GRU показывают более усредненные значения, модель GARCH имеет больший разброс. Построим таблицу ошибок для каждой модели и определим какая модель лучше.

В таблице 1 приведён сравнительный анализ моделей.

Таблица 1 – Сравнительный анализ моделей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель | Ошибки | | | Время обучения и предсказания / секунд |
| MSE | MAE | RMSE |
| Тривиальная | 8.40 | 1.98 | 2.89 | - |
| GARCH | 5.60 | 1.57 | 2.36 | 40 |
| LSTM | 5.17 | 1.39 | 2.27 | 23 |
| GRU | 5.04 | 1.43 | 2.46 | 16 |

Исходя из полученных данных можно сделать вывод: каждая модель показала результат лучше тривиальной на 30%.

Каждая модель имеет свои особенности, однако для дальнейшего использования рекомендовано использовать модель LSTM, так как она имеет более широкий спектр настроек и возможность использовать несколько входных данных на вход.

3.5.2 Интерпретация полученных результатов

Рассмотрим полученные данные и определим критерии волатильности.

На рисунке 29 представлена разбивка предсказанных значений для модели LSTM.

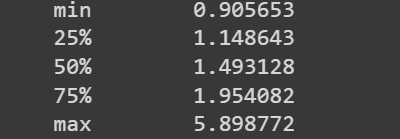


Рисунок 29 – Разбивка предсказанных значений для модели LSTM

В таблице 2 проведена интерпретация показателей модели LSTM.

Таблица 2 – Интерпретация показателей модели LSTM

|  |  |
| --- | --- |
| Модель | Показатель волатильности, V |
|
| Очень малая волатильность | V < 1.15 |
| Малая волатильность | 1.95 > V > 1.15 |
| Высокая волатильность | 2 > V > 1.95 |
| Очень высока волатильность | V > 2 |

# ЗАКЛЮЧЕНИЕ

В рамках данной работы были построены модели GARCH, LSTM и GRU на примере прогнозирования волатильности фондового индекса Российского рынка RTS.

В результате проведенного исследования было установлено, что все модели имеют высокую точность прогнозирования относительно тривиальной модели. Однако, модель LSTM оказалась более точной и имеет среднее время выполнения. Плюсом модели LSTM является возможность дальнейшего масштабирования.

В заключении можно отметить, что применение методов машинного обучения для прогнозирования показателей волатильности является эффективным инструментом, который может быть использован для принятия решений в инвестиционной деятельности.

# Приложение А

(обязательное)

Исходный код программы

# Библиотеки для работы с гугл диском

from google.colab import drive

drive.mount('/content/drive')

# Библиотеки для работы с данным

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

# Библиотека для визуализации данных

import matplotlib.pyplot as plt

import seaborn as sns

# Библиотеки для построение модели

from keras.models import Sequential

from keras.layers import Dense, Dropout, LSTM, GRU

from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

from keras.optimizers import Nadam

# Библиотека учета времени

import time

# Библиотеки для расчета ошибок

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import mean\_absolute\_error

# Загрузка данных

data = pd.read\_csv('/content/drive/MyDrive/kr/RTS.csv', sep=',')

data

# Обработка данных

data = data.drop(['<TICKER>', '<PER>', '<TIME>'], axis=1) # Удаление столбцов которые не несут полезной информации

data = data.rename(columns={'<DATE>': 'date', '<CLOSE>': 'close', '<OPEN>': 'open', '<HIGH>': 'high', '<LOW>': 'low', '<VOL>': 'volume'}) # Перереименуем столбцы для удобства

data['date'] = pd.to\_datetime(data['date'], format='%Y%m%d') # Преобразуем столбец 'date' в объекты datetime и сделаем его индексом , сортируем индекс

data = data.set\_index('date').sort\_index()

data

# Визуализация данных

plt.figure(figsize=(30,7))

plt.plot(data.index, data['open'].values, label = 'График цены открытия RTS', color = 'green')

plt.plot(data.index, data['close'].values, label = 'График цены закрытия RTS', color = 'red')

plt.plot(data.index, data['high'].values, label = 'График максимальной цены RTS', color = 'orange')

plt.plot(data.index, data['low'].values, label = 'График минимальной цена RTS', color = 'brown')

plt.xlabel('Дата')

plt.ylabel('Цена')

plt.legend()

plt.figure(figsize=(30,7))

plt.plot(data.index, data['volume'].values, label = 'RTS график объема', color = 'red')

plt.xlabel('Дата')

plt.ylabel('Объем')

plt.legend()

plt.show()

# Преобразование данные в абсолютное процентное изменение

norm\_data = abs(data[["open", "high", "low", "close", "volume"]].diff()/ data[["open", "high", "low", "close", "volume"]].shift(1)\*100).dropna()

norm\_data

# Визуализация данных

plt.figure(figsize=(30,7))

plt.plot(norm\_data.index, norm\_data['open'].values, label = 'Абсолютные процентные изменения цены открытия RTS', color = 'green')

plt.plot(norm\_data.index, norm\_data['close'].values, label = 'Абсолютные процентные изменения цены закрытия RTS', color = 'red')

plt.plot(norm\_data.index, norm\_data['high'].values, label = 'Абсолютные процентные изменения максимальной цены RTS', color = 'orange')

plt.plot(norm\_data.index, norm\_data['low'].values, label = 'Абсолютные процентные изменения минимальной цены RTS', color = 'brown')

plt.xlabel('Дата')

plt.ylabel('Цена')

plt.legend()

plt.figure(figsize=(30,7))

plt.plot(norm\_data.index, norm\_data['volume'].values, label = 'RАбсолютные процентные изменения объема', color = 'red')

plt.xlabel('Дата')

plt.ylabel('Объем')

plt.legend()

plt.show()

# Вычислим целевую переменную: скользящее стандартное отклонение за n периодов

volatility = abs(data['close'].diff() / data['close'].shift(1)\*100)

volatility.name = "volatility"

input\_data = pd.concat([norm\_data, volatility], axis=1).dropna()

input\_data['returns'] = data['close'].diff() / data['close'].shift(1)\*100

input\_data

# Визуализация данных

plt.figure(figsize=(20,5))

plt.plot(input\_data.index, input\_data['open'].values, label = 'График волатильности цены открытия RTS', color = 'green')

plt.plot(input\_data.index, input\_data['close'].values, label = 'График волатильности цены закрытия RTS', color = 'red')

plt.plot(input\_data.index, input\_data['high'].values, label = 'График волатильности максимальной цены RTS', color = 'orange')

plt.plot(input\_data.index, input\_data['low'].values, label = 'График волатильности минимальной цена RTS', color = 'brown')

plt.xlabel('Дата')

plt.ylabel('Минимальная цена')

plt.legend()

plt.figure(figsize=(20,5))

plt.plot(input\_data.index, input\_data['volume'].values, label = 'RTS волатильности объема', color = 'red')

plt.xlabel('Дата')

plt.ylabel('Объем')

plt.legend()

plt.show()

# Создание фигуры с 3 столбцами и 2 строками

fig, axes = plt.subplots(2, 4, figsize=(20, 10))

# Создание счетчика для отслеживания текущего подзаголовка

counter = 0

# Перебор всех столбцов фрейма данных

for column in input\_data.columns:

    # Создание прямоугольной диаграммы для каждого столбца

    ax = axes[counter // 4, counter % 3]

    sns.boxplot(input\_data[column], ax=ax)

    ax.set\_title(f"Выбросы в {column}")

    counter += 1

plt.show()

input\_data['volatility\_clip']= input\_data['volatility'].clip(lower=-7, upper=7)

input\_data['open']= input\_data['open'].clip(lower=0, upper=7)

input\_data['high']= input\_data['high'].clip(lower=0, upper=7)

input\_data['low']= input\_data['low'].clip(lower=0, upper=7)

input\_data['close']= input\_data['close'].clip(lower=0, upper=7)

input\_data['volume']= input\_data['volume'].clip(lower=0, upper=300)

input\_data['returns']= input\_data['returns'].clip(lower=-7, upper=7)

input\_data

scaler = MinMaxScaler(feature\_range=(0, 7))

input\_data['volume'] = scaler.fit\_transform(input\_data[['volume']])

# Создание фигуры с 3 столбцами и 2 строками

fig, axes = plt.subplots(3, 3, figsize=(30, 15))

# Создание счетчика для отслеживания текущего подзаголовка

counter = 0

# Перебор всех столбцов фрейма данных

for column in input\_data.columns:

    # Создание прямоугольной диаграммы для каждого столбца

    ax = axes[counter // 3, counter % 3]

    sns.boxplot(input\_data[column], ax=ax)

    ax.set\_title(f"Выбросы в {column}")

    counter += 1

plt.show()

# Визуализируем результат

plt.figure(figsize=(30,5))

plt.plot(input\_data.index, input\_data['open'].values, label = 'График волатильности цены открытия RTS', color = 'green')

plt.plot(input\_data.index, input\_data['close'].values, label = 'График волатильности цены закрытия RTS', color = 'red')

plt.plot(input\_data.index, input\_data['high'].values, label = 'График волатильности максимальной цены RTS', color = 'orange')

plt.plot(input\_data.index, input\_data['low'].values, label = 'График волатильности минимальной цена RTS', color = 'brown')

plt.xlabel('Дата')

plt.ylabel('Минимальная цена')

plt.legend()

plt.figure(figsize=(30,5))

plt.plot(input\_data.index, input\_data['volume'].values, label = 'RTS график волатильности объема', color = 'red')

plt.xlabel('Дата')

plt.ylabel('Объем')

plt.legend()

plt.show()

plt.figure(figsize=(30,5))

plt.plot(input\_data.index, input\_data['volatility\_clip'].values, label = 'RTS график абсолюной волатильности', color = 'green')

plt.xlabel('Дата')

plt.ylabel('Объем')

plt.legend()

plt.show()

input\_data.hist(color='green',figsize=(30,5))

plt.show()

# Создадим тренировочный и тестовые выборки данных

X = input\_data[['open', 'high', 'low', 'close', 'volume']] # Входные данные

Y = input\_data['volatility\_clip'] # Выходные данные

# Функция скользящего окна

def split\_data(X, Y, WINDOW, FUTURE\_STEP):

    \_X = X.dropna()

    \_Xs = []

    \_Ys = []

    for i in range(\_X.shape[0] - WINDOW - FUTURE\_STEP + 1):

        cur\_X = \_X.values[i:i + WINDOW]

        \_Xs.append(cur\_X)

        y\_pointer = i + WINDOW + FUTURE\_STEP - 1

        cur\_Y = Y.values[y\_pointer]

        \_Ys.append(cur\_Y)

    return np.array(\_Xs), np.array(\_Ys)

WINDOW = 5

FUTURE\_STEP = 1

x, y = split\_data(X, Y, WINDOW, FUTURE\_STEP)

# Устанавливаем размерность выборок 90% train, 10% test

x\_train, x\_test = x[:int(len(x) \* .9)], x[int(len(x) \* .9):]

y\_train, y\_test = y[:int(len(y) \* .9)], y[int(len(y) \* .9):]

# Построение тривиальной модели

X2 = input\_data[['open', 'high', 'low', 'close']] # Входные данные

Y2 = input\_data['volatility'] # Выходные данные

x2, y2 = split\_data(X2, Y2, WINDOW, FUTURE\_STEP)

# Устанавливаем размерность выборок 90% train, 10% test

xr\_train, xr\_test = x2[:int(len(x) \* .9)], x2[int(len(x) \* .9):]

yr\_train, yr\_test = y2[:int(len(y) \* .9)], y2[int(len(y) \* .9):]

result = pd.DataFrame(columns=["real", 'trivial\_shift'])

result['real'] = yr\_test

result['trivial\_shift'] = result['real']

result['trivial\_shift'] = result['trivial\_shift'].shift(-1).fillna(np.mean(result['trivial\_shift']))

Trivial\_MSE\_shift = mean\_squared\_error(result['real'], result['trivial\_shift'])

Trivial\_MAE\_shift = mean\_absolute\_error(result['real'], result['trivial\_shift'])

Trivial\_RMSЕ\_shift = np.sqrt(Trivial\_MSE\_shift)

print( "Тривиальня сдвиг MSE:", Trivial\_MSE\_shift)

print( "Тривиальня сдвиг MAE:", Trivial\_MAE\_shift)

print( "Тривиальня сдвиг RMSE:", Trivial\_RMSЕ\_shift)

!pip install arch

# Подготовка данных для модели GARCH

xg, yg = split\_data(input\_data['returns'], input\_data['returns'], WINDOW, FUTURE\_STEP)

xg\_train, xg\_test = xg[:int(len(xg) \* .9)], xg[int(len(xg) \* .9):]

yg\_train, yg\_test = yg[:int(len(yg) \* .9)], yg[int(len(yg) \* .9):]

GARCHstart\_time = time.time() # Начальная точка отсчета

from arch import arch\_model

forecat2 = []

for i in xg\_test:

  print(i)

  model = arch\_model(i, vol = 'GARCH', p=1, q = 1).fit()

  forecat2.append(model.forecast(horizon = 1).variance)

GARCHend\_time= time.time() # Конечная точка отсчета

GARCHtime = GARCHend\_time - GARCHstart\_time # Расчет времени выполнения

print("Время, затраченное на компиляцию модели GARCH: {} секунды".format(GARCHtime))

forecast2\_array = np.array(forecat2)

forecast2\_reshaped = forecast2\_array.reshape((351, 5))

df\_forecast2 = pd.DataFrame(forecast2\_reshaped, columns=['Step 1', 'Step 2', 'Step 3', 'Step 4', 'predict'])

GARCHpred\_df = pd.DataFrame(np.column\_stack([result['real'], df\_forecast2['predict']]), columns=["real", "GARCHpred"])

GARCHpred\_df['GARCHpred'] = np.sqrt(GARCHpred\_df['GARCHpred'])

cc = result['real']

# Расчет MSE MAE RMSE прогноза

GARCH\_MSE = mean\_squared\_error(result['real'], GARCHpred\_df['GARCHpred'])

GARCH\_MAE = mean\_absolute\_error(result['real'], GARCHpred\_df['GARCHpred'])

GARCH\_RMSE = np.sqrt(GARCH\_MSE)

print("Trivial MSE:", Trivial\_MSE\_shift, "GARCH MSE:", GARCH\_MSE)

print("Trivial MAE:", Trivial\_MAE\_shift, "GARCH MAE:", GARCH\_MAE)

print("Trivial RMSE:", Trivial\_RMSЕ\_shift, "GARCH RMSE:", GARCH\_RMSE)

GARCHpred\_df.plot(figsize=(30,5))

# Построение модели LSTM

\_data\_shape = (5, 5) # Указываем размер входных данных(окно, количество данных)

model = Sequential()

model.add(LSTM(64 , activation='linear', input\_shape=\_data\_shape, return\_sequences=False))

model.add(Dropout(0.5))

model.add(Dense(16, kernel\_initializer='uniform',activation='relu'))

model.add(Dense(1, kernel\_initializer='uniform',activation='linear'))

opt = Nadam(learning\_rate=0.002, clipnorm = .1)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.9, patience=3, min\_lr=0.000001, verbose=1)

checkpointer = ModelCheckpoint(monitor='val\_loss', filepath="model\_returns.hdf5", verbose=1, save\_best\_only=True)

model.compile(optimizer=opt, loss='mse')

# Обучение модели

LSTMstart\_time = time.time() # Начальная точка отсчета

history = model.fit(

    x\_train,

    y\_train,

    batch\_size=16,

    epochs=10,

    validation\_split=0.1,

    callbacks=[reduce\_lr, checkpointer],

    verbose=1)

y\_pred = model.predict(x\_test)

LSTMend\_time= time.time() # Конечная точка отсчета

LSTMtime = LSTMend\_time - LSTMstart\_time # Расчет времени выполнения

print("Время, затраченное на компиляцию модели LSTM: {} секунды".format(LSTMtime))

# Сделаем предсказания на основе тестовых данных и сравним результат

pred\_df = pd.DataFrame(np.column\_stack([result['real'], y\_pred]), columns=["Real", "LSTM\_pred"])

# Выведем график

pred\_df.plot(figsize=(30,5))

plt.show()

# Расчет MSE MAE RMSE прогноза

LSTM\_MSE = mean\_squared\_error(result['real'], pred\_df['LSTM\_pred'])

LSTM\_MAE = mean\_absolute\_error(result['real'], pred\_df['LSTM\_pred'])

LSTM\_RMSE = np.sqrt(LSTM\_MSE)

print("Trivial MSE:", Trivial\_MSE\_shift, "LSTM MSE:", LSTM\_MSE)

print("Trivial MAE:", Trivial\_MAE\_shift, "LSTM MAE:", LSTM\_MAE)

print("Trivial RMSE:", Trivial\_RMSЕ\_shift, "LSTM RMSE:", LSTM\_RMSE)

# Построение модели GRU

GRUmodel = Sequential()

GRUmodel.add(GRU(64, activation='linear', input\_shape=\_data\_shape, return\_sequences=False))

GRUmodel.add(Dropout(0.5))

GRUmodel.add(Dense(16, kernel\_initializer='uniform',activation='relu'))

GRUmodel.add(Dense(1, kernel\_initializer='uniform',activation='linear'))

opt = Nadam(learning\_rate=0.002, clipnorm = .1)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.9, patience=3, min\_lr=0.000001, verbose=1)

checkpointer = ModelCheckpoint(monitor='val\_loss', filepath="model\_returns.hdf5", verbose=1, save\_best\_only=True)

GRUmodel.compile(optimizer=opt, loss='mse')

# Обучение модели GRU

GRUstart\_time = time.time() # Начальное время отсчета

GRUhistory = GRUmodel.fit(

    x\_train,

    y\_train,

    batch\_size=16,

    epochs=10,

    validation\_split=0.1,

    callbacks=[reduce\_lr, checkpointer],

    verbose=1)

GRUy\_pred = GRUmodel.predict(x\_test)

GRUend\_time= time.time() # Конечное время отсчета

GRUtime = GRUend\_time - GRUstart\_time # Расчет времени обучение модели GRU

print("Время, затраченное на компиляцию модели GRU: {} секунды".format(GRUtime))

# Сделаем предсказания на основе тестовых данных и сравним результат

GRUpred\_df = pd.DataFrame(np.column\_stack([result['real'], GRUy\_pred]), columns=["real", "GRUpred"])

# Выведем график

GRUpred\_df.plot(figsize=(30,7))

plt.show()

# Расчет MSE MAE RMSE прогноза

GRU\_MSE = mean\_squared\_error(result['real'],GRUpred\_df['GRUpred'])

GRU\_MAE = mean\_absolute\_error(result['real'], GRUpred\_df['GRUpred'])

GRU\_RMSE = np.sqrt(GRU\_MSE)

print("Trivial MSE:", Trivial\_MSE\_shift, "GRU MSE:", GRU\_MSE)

print("Trivial MAE:", Trivial\_MAE\_shift, "GRU MAE:", GRU\_MAE)

print("Trivial RMSE:", Trivial\_RMSЕ\_shift, "GRU RMSE:", GRU\_RMSE)

all\_df = pd.DataFrame(columns=["real", "Garch", 'LSTM', 'GRU'])

all\_df['real'] = result['real']

all\_df['Garch'] = np.sqrt(df\_forecast2['predict'])

all\_df['LSTM'] = pred\_df['LSTM\_pred']

all\_df['GRU'] = GRUpred\_df['GRUpred']

# all\_df['trivial'] = result['trivial\_shift']

all\_df.plot(figsize=(50,15))

print("Время, затраченное на компиляцию модели GARCH: {} секунды".format(GARCHtime))

print("Время, затраченное на компиляцию модели LSTM: {} секунды".format(LSTMtime))

print("Время, затраченное на компиляцию модели GRU: {} секунды".format(GRUtime))

print("Trivial MSE:", Trivial\_MSE\_shift, "GRU MSE:", GRU\_MSE)

print("Trivial MAE:", Trivial\_MAE\_shift, "GRU MAE:", GRU\_MAE)

print("Trivial RMSE:", Trivial\_RMSЕ\_shift, "GRU RMSE:", GRU\_RMSE)

print("Средняя квадратическая ошибка MSE :", "LSTM MSE:", LSTM\_MSE, "GRU MSE:", GRU\_MSE,"GARCH MSE:", GARCH\_MSE)

print("Средняя абсолютная ошибка MAE:", "LSTM MAE:", LSTM\_MAE, "GRU MAE:", GRU\_MAE,"GARCH MAE:", GARCH\_MAE)

print("Среднеквадратическая ошибка RMSE:", "LSTM RMSE:", LSTM\_RMSE, "GRU RMSE:", GRU\_RMSE,"GARCH RMSE:", GARCH\_RMSE)

all\_df