Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет \_Электроники и вычислительной техники\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра \_\_ Электронно-вычислительные машины и системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

##### **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе (проекту)**

по дисциплине \_\_Системы обработки больших данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

на тему\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент\_\_\_\_\_Бондаренко Артём Геннадьевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

##### (фамилия, имя, отчество)

Группа\_\_\_\_\_\_САПР-1.3\_\_\_\_\_\_\_

Руководитель работы (проекта) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_П.Д. Кравченя\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата подписания) (инициалы и фамилия)

Волгоград 2023 г.

Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет \_\_\_ Электроники и вычислительной техники\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление (специальность)\_\_Информатика и вычислительная техника\_\_\_\_\_\_  
Кафедра \_\_ Электронно-вычислительные машины и системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дисциплина\_\_ Системы обработки больших данных \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  | Утверждаю  Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | «\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |

##### **Задание**

##### **на курсовую работу (проект)**

Студент\_\_\_\_ Бондаренко Артём Геннадьевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

##### (фамилия, имя, отчество)

Группа\_\_\_\_\_ САПР-1.3\_\_\_\_\_\_\_\_\_\_

1. Тема: \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утверждена приказом от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г. № \_\_\_\_\_\_\_\_\_

2. Срок представления работы (проекта) к защите «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

3. Содержание расчетно-пояснительной записки: 1. Пояснительная записка;\_\_\_ \_\_\_\_\_\_\_2.Задание на курсовую работу;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_3.Содержание;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_4.Введение;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_5. Разведочный анализ данных с помощью PySpark; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_6. Машинное обучение на больших данных;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_7.Заключение;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_**

\_\_\_\_\_\_\_8.Список используемых источников;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_9.Приложения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Перечень графического материала: \_4.1 Рис. 1. Типы данных в датасете \_\_\_\_

\_\_\_\_\_4.2 Табл. 1. Типы признаков в датасете\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.3 Рис. 2 - 3. Количество пропущенных значений в датасете (до и после\_ удаления)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.4 Рис. 4, 7. Статистические данные для отображения количества\_\_\_\_\_\_ оставшихся строк в датасете (до и после удаления)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.5 Рис. 5 - 6. Коробчатые диаграммы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.6 Рис. 8 - 10. Статистические показатели признаков\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.7 Рис. 11. Гистограммы распределения количественных\_\_\_\_\_\_\_\_\_\_\_\_\_\_ признаков\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.8 Рис. 12. Круговая диаграмма бинарного признака\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.9 Рис. 13 - 17. Диаграммы рассеиваний признаков\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.10 Табл. 2. Матрица корреляции\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.11 Рис. 18. Результат вывода значений из обработанного\_\_\_\_\_\_\_\_\_\_\_\_\_ датасета\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.12 Рис. 19, 22. Предсказания на основе тестовых\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.13 Рис. 20. Признаки, использованные в задачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ классификации\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.14 Рис. 21. Количество значений к колонке «label»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.15 Табл. 3 - 4. Метрики модели регрессии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.16 Табл. 5 - 6. Метрики модели классификации\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_4.17 Табл. 7 - 8. Анализ метрик\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Дата выдачи задания «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г.

Руководитель работы (проекта)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_П.Д. Кравченя\_\_\_

подпись, дата инициалы и фамилия

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата инициалы и фамилия

Содержание:

[Введение 5](#_Toc153765891)

[1. Разведочный анализ данных с помощью PySpark 6](#_Toc153765892)

[1.1 Постановка задачи 6](#_Toc153765893)

[1.2 Описание датасета 6](#_Toc153765894)

[1.3 Типы признаков в датасете 7](#_Toc153765895)

[1.4 Выявление пропущенных значений и их устранение 10](#_Toc153765896)

[1.5 Определение выбросов и их устранение 12](#_Toc153765897)

[1.6 Расчет статистических показателей признаков 16](#_Toc153765898)

[1.7 Визуализацией распределения наиболее важных признаков 21](#_Toc153765899)

[1.8 Корреляций между признаками 29](#_Toc153765900)

[1.9 Выводы 31](#_Toc153765901)

[2. Машинное обучение на больших данных 33](#_Toc153765902)

[2.1 Постановка задачи 33](#_Toc153765903)

[2.2 Применение алгоритмов подготовки данных и машинного обучения 33](#_Toc153765904)

[2.3 Задача регрессии – GradientBoostingMachine 35](#_Toc153765905)

[2.4 Задача классификации – LogisticRegression 39](#_Toc153765906)

[2.5 Анализ результатов 46](#_Toc153765907)

[2.6 Вывод 47](#_Toc153765908)

[Заключение 48](#_Toc153765909)

[Список используемых источников 49](#_Toc153765910)

[Приложение А 51](#_Toc153765911)

[Приложение Б 58](#_Toc153765912)

**Введение**

В настоящее время системы обработки и анализа больших данных широко используются в различных областях, таких как бизнес, наука, здравоохранение, финансы, телекоммуникации, интернет вещей и многие другие. Эти системы позволяют организациям извлекать ценные знания из огромных объемов данных, что в свою очередь помогает принимать обоснованные решения и выявлять скрытые закономерности.

Важность, как и **актуальность**, анализа данных и Data Science в современном мире огромна. Они помогают организациям принимать обоснованные решения, оптимизировать бизнес-процессы, предсказывать тренды и поведение клиентов. Без анализа данных и Data Science невозможно представить себе успешное функционирование таких областей, как финансы, маркетинг, медицина и многие другие.

Apache Spark - это высокопроизводительный кластерный фреймворк для обработки больших данных. PySpark - это API для работы с Apache Spark, предназначенное для использования с языком программирования Python. PySpark позволяет использовать все преимущества Apache Spark, используя удобный и мощный Python.

PySpark предоставляет набор инструментов для машинного обучения, включая модуль MLlib, который предлагает широкий выбор алгоритмов машинного обучения и инструменты для обработки признаков.

**Целью** работы является изучение инстумента PySpark и применение навыков для осуществление разведочного анализа и машинного обучения.

Для реализации цели были решены следующие **задачи**:

* Провести разведочный анализ данных с помощью PySpark;
* Разобрать машинное обучение на больших данных.

1. **Разведочный анализ данных с помощью PySpark**
   1. **Постановка задачи**

Цель и задачи работы:

1. Познакомиться с понятием «большие данные» и способами их обработки;
2. Познакомиться с инструментом Apache Spark и возможностями, которые он предоставляет для обработки больших данных.
3. Получить навыки выполнения разведочного анализа данных использованием pyspark.

Необходимо провести разведочный анализ датасета с определением:

* типов признаков в датасете;
* пропущенных значений и их устранением;
* выбросов и их устранением;
* расчетом статистических показателей признаков (средних, квартилей и т.д.);
* визуализацией распределения наиболее важных признаков;
* корреляций между признаками.
  1. **Описание датасета**

Архив с датасетами твитов общественного мнения и политики из Твиттера был выбран, т.к. является индивидуальным для моего варианта.

Ссылка – https://www.kaggle.com/datasets/paultimothymooney/twitter-election-data-archives

В качестве датасета, на основе которого проводился разведочный анализ, был выбран файл формата CSV под названием «iran\_201906\_2\_tweets\_csv\_hashed.csv» находящийся в папке «June 2019». Размер файла составляет 1.40 GB.

Данный датасет содержит порядка 7417013 строк, которые описывают каждый выложенный пост в сервисе «Twitter» за июнь 2019 года. В датасете было обнаружено 31 колонка. Из них, для дальнейшего анализа, было решено оставить 10 колонн.

Датасет состоит из таких колонок, как:

* 1. Идентификатор твита. Является уникальным идентификатором для каждого поста.
  2. Идентификатор пользователя. Является уникальным идентификатором для каждого пользователя.
  3. Количество подписчиков у пользователя.
  4. Количество подписок.
  5. Язык поста.
  6. Ретвит или оригинальный пост.
  7. Количество цитирований поста.
  8. Количество ответов поста.
  9. Количество лайков поста.
  10. Количество ретвитов поста.
  11. **Типы признаков в датасете**

Необходимо определить типы признаков в датасете для дальнейшего анализа и использования данных признаков для выявления пропущенных значений, выбросов, статистических данных, визуализации диаграмм и определения корреляции.

В датасете схемы обнаружено не было и пришлось вручную удалять и изменять типы признаков вручную, основываясь на названии и просмотром данных нескольких строк данных.

df = df.drop("user\_profile\_description", "account\_creation\_date", "user\_reported\_location", "account\_language", "tweet\_text", "tweet\_time", "tweet\_client\_name", "in\_reply\_to\_userid", "in\_reply\_to\_tweetid", "quoted\_tweet\_tweetid", "retweet\_userid", "retweet\_tweetid", "latitude", "longitude", "user\_mentions", "poll\_choices", "hashtags", "urls", "user\_screen\_name", "user\_display\_name", "user\_profile\_url")

df = df.withColumn("tweetid", col("tweetid").cast(LongType()))

df = df.withColumn("userid", col("userid").cast(LongType()))

df = df.withColumn("follower\_count", col("follower\_count").cast(IntegerType()))

df = df.withColumn("following\_count", col("following\_count").cast(IntegerType()))

df = df.withColumn("quote\_count", col("quote\_count").cast(IntegerType()))

df = df.withColumn("reply\_count", col("reply\_count").cast(IntegerType()))

df = df.withColumn("like\_count", col("like\_count").cast(IntegerType()))

df = df.withColumn("retweet\_count", col("retweet\_count").cast(IntegerType()))

df = df.withColumn("is\_retweet", col("is\_retweet").cast("boolean"))

print('Data overview')

df.printSchema()

print('Columns overview')

pd.DataFrame(df.dtypes, columns = ['Column Name','Data type'])

После применения алгоритма, была сформирована схема всех колонок датасета, а также и таблица с типами колонок (рис.1).

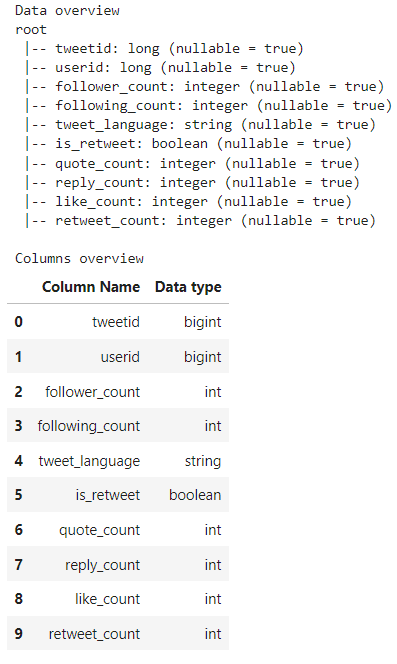


Рисунок 1. Типы данных в датасете

В ходе анализа датасета были выявлены следующие типы признаков в датасете, описанных в табл. 1.

Таблица 1. Типы признаков в датасете

|  |  |  |
| --- | --- | --- |
| Название колонки | Тип данных | Тип признака |
| tweetid | long | Категориальный |
| userid | long | Категориальный |
| follower\_count | integer | Количественный |
| following\_count | integer | Количественный |
| tweet\_language | string | Категориальный |
| is\_retweet | boolean | Бинарный |
| quote\_count | integer | Количественный |
| reply\_count | integer | Количественный |
| like\_count | integer | Количественный |
| retweet\_count | integer | Количественный |

В дальнейшем для осуществления обучения модели машинного обучения колонки количественного типа признаков будут преобразованы в тип данных «double».

* 1. **Выявление пропущенных значений и их устранение**

Чтобы корректно проанализировать данные, важно удостовериться, что в наборе данных отсутствуют пропущенные значения.

В ходе анализа пропущенных значений данные были разделены на 2 группы: строковые и числовые колонки, где:

* Строковые колонки проверялись на наличие значений None и null.
* Числовые колонки проверялись на наличие значений None, NaN и null.

string\_columns = ['tweet\_language', 'is\_retweet']

numeric\_columns = ['tweetid', 'userid', 'follower\_count', 'following\_count', "quote\_count", "reply\_count", "like\_count", "retweet\_count"]

def view\_missing\_values\_in\_columns(string\_columns, numeric\_columns):

missing\_values = {}

for index, column in enumerate(df.columns):

if column in string\_columns:

missing\_count = df.filter(col(column).eqNullSafe(None) | col(column).isNull()).count()

else:

missing\_count = df.where(col(column).isin([None,np.nan]) | col(column).isNull()).count()

missing\_values.update({column:missing\_count})

return missing\_values

pd.DataFrame.from\_dict([view\_missing\_values\_in\_columns(string\_columns, numeric\_columns)])

После применения функции обнаружения пропущенных значений была сформирована таблица со значениями (рис. 2), который отображает количество пропущенных данных.

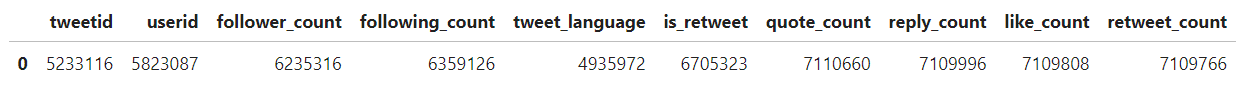


Рисунок 2. Количество пропущенных значений в датасете

После анализа пропущенных значений были получены такие результаты:

* tweetid: 5233116
* userid: 5823087
* follower\_count: 6235316
* following\_count: 6359126
* tweet\_language: 4935972
* is\_retweet: 6705323
* quote\_count: 7110660
* reply\_count: 7109996
* like\_count: 7109808
* retweet\_count: 7109766

df = df.dropna()

print('Устранение пропущенных значений прошло успешно')

pd.DataFrame.from\_dict([view\_missing\_values\_in\_columns(string\_columns, numeric\_columns)])

Строки с пропущенными значениями решено было удалить из датасета. А после проверить, остались ли пропущенные значения в датасете, вызвав функцию проверки снова (рис.3).

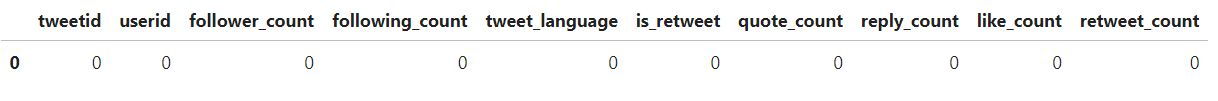


Рисунок 3. Количество пропущенных значений после удаления этих значений

Количество строк после удаления пропущенных значений – 199496 (рис.4). Это было выведено с помощью функции describe() - которая и выводит статистические показатели, преобразовав в таблицу с помощью метода toPandas().

df.select(df.follower\_count, df.following\_count, df.quote\_count, df.reply\_count, df.like\_count, df.retweet\_count).describe().toPandas()

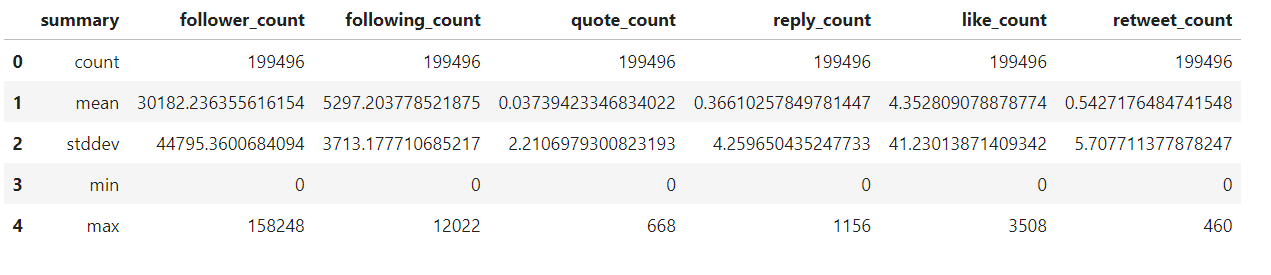


Рисунок 4. Статистические данные для отображения количества оставшихся строк в датасете

* 1. **Определение выбросов и их устранение**

Осуществим обнаружение и удаление выбросов методом межквартильного интервала (Нахождение 25% и 75% квартилей, межквартильного диапазона):

Данные колонок "quote\_count", "reply\_count", "like\_count", "retweet\_count" решено было не обрабатывать, т.к. во всех датасетах, которые имеются в архиве, приводит к тому, что все данные колонок удаляются и остаются нулевые значения, а данные колонки будут нужны для обучения модели и формирования графиков. Это связанно с тем, что большая часть постов не имеют никаких метрических реакций других пользователей.

Исходный код метода межквартильного интервала, с помощью которого определяются и удаляются выбросы.

outlier\_columns = ['follower\_count', 'following\_count']

for column in outlier\_columns:

Q1 = df.approxQuantile(column, [0.25], relativeError=0)[0]

Q3 = df.approxQuantile(column, [0.75], relativeError=0)[0]

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

df = df.filter(col(column).between(lower\_bound, upper\_bound))

# Передаем необходимые значения на драйвер для визуализации

boxplot\_data = df.select(column).collect()

values = [row[column] for row in boxplot\_data]

# Визуализация выбросов с помощью boxplot

plt.figure(figsize=(7, 5))

plt.boxplot(values, vert=False, widths=0.7, patch\_artist=True, showmeans=False, autorange=True)

plt.title('Boxplot с выбросами для {}'.format(column))

plt.xlabel('Values')

plt.show()

df.select(['follower\_count', 'following\_count', 'quote\_count', 'reply\_count', 'like\_count', 'retweet\_count']).describe().toPandas()

Метод межквартильного интервала представляет собой формирование 25 процентиля из 100 и формирование 75 процентиля из 100, данные формируются максимально точно, не допуская ошибок, за это отвечает параметр relativeError. После формируется межквартильный размах, который включает себя разность 75 процентиля от 25. По готовой формуле формируются нижние и верхние границы наблюдаемых значений, которые не будут считаться выбросами. После установления границ, с помощью метода filter, алгоритм проверяет данные и те, что находятся за пределами нижней и верхней границы, удаляются.

После применения метода передаем все полученные данные на драйвер и формируем коробчатые диаграммы (рис. 5 – 6) по значениям, полученных при использовании метода межквартильного интервала.

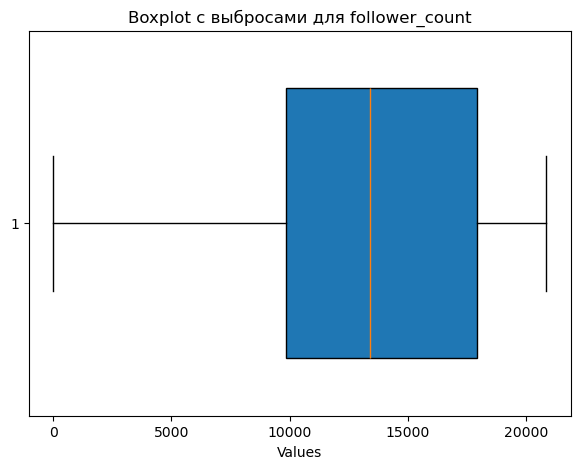


Рисунок 5. Ящик с усами для колонки «follower\_count»

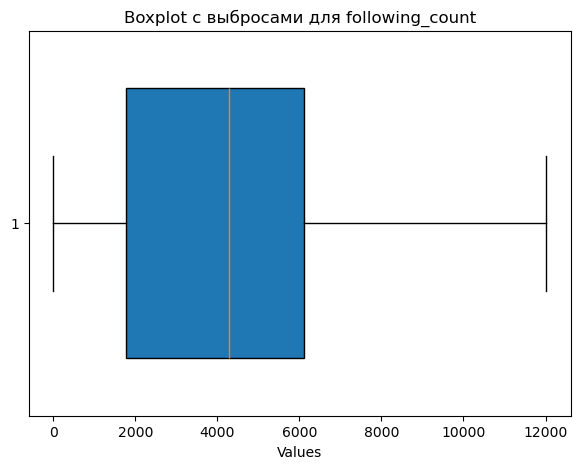


Рисунок 6. Ящик с усами для колонки «following\_count»

А после формирования коробчатых диаграмм выводим статистические данные колонок, для установления количества строк в датасете. Количество строк после выявления и удаления выбросов – 174266 (рис.7).

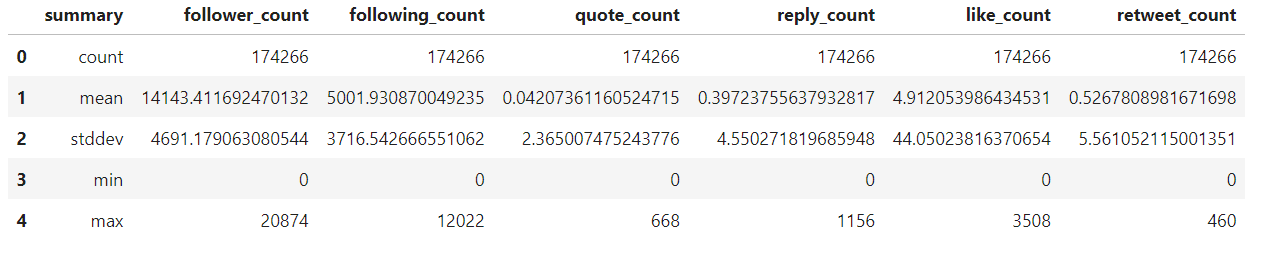


Рисунок 7. Статистические данные для отображения количества оставшихся строк в датасете после удаления выбросов

Из рисунка 7 видно, что количество строк уменьшилось в сравнении с данными с рис. 4. Рисунки 5-6 демонстрируют то, что в колонках follower\_count и following\_count не обнаружено выбросов, т.к. они были удалены.

* 1. **Расчет статистических показателей признаков**

Для реализации этого пункта было решено разобрать категориальные, бинарные и количественные признаки.

Обработаем колонки количественных признаков, а также по одной колонке категориального и одной колонки бинарного признака. Начнём с количественных признаков:

Статистические показатели обширны, мной были выбраны несколько основные из них:

1. Среднее (Mean): Обычно является первым показателем, который рассчитывается для числовых данных. Среднее значение предоставляет информацию о центре распределения данных.

2. Медиана (Median): Это значение, которое делит ваш набор данных на две равные части. Она устойчива к выбросам в данных и используется для оценки "среднего" значения, если данные имеют асимметричное распределение.

3. Квартили (Quartiles): Квартили представляют собой три точки данных, которые делят упорядоченный набор данных на четыре равные части. 25-й квартиль - это значение, которое меньше или равно 25% данных, а 75-й квартиль - это значение, которое меньше или равно 75% данных.

4. Мода (Mode): Значение, которое встречается наиболее часто в вашем наборе данных. Мода полезна для определения наиболее распространенных значений в категориальных данных.

5. Стандартное отклонение (Standard Deviation): Это мера разброса данных относительно их среднего значения. Она предоставляет информацию о том, насколько разнообразны значения в вашем наборе данных.

6. Минимум (Min): Минимальный показатель колонки.

7. Максимум (Max): Максимальный показатель колонки.

8. Диапазон значений (Range): Простое измерение, которое показывает разницу между самым большим и самым маленьким значениями в наборе данных.

9. Квантили (Quantiles): Квантили обобщают понятие квартиля. В то время как квартили делят данные на четыре равные части, квантили позволяют делить данные на более произвольное количество частей.

def calculate\_statistics(df, columns):

statistics = {'Statistic': ['Mean', 'Median', 'Mode', 'Standard Deviation', 'Min', 'Max', 'Range', '25th Quantile', '75th Quantile', '95th Quartiles']}

for column in columns:

mode\_column = df.groupBy(column).count().sort(F.col("count").desc()).select(column).limit(1).collect()[0][0]

column\_values = [

df.agg({column: "avg"}).collect()[0][0], # среднее значение

df.approxQuantile(column, [0.5], 0.001)[0], # медиана

mode\_column, # мода

df.agg({column: "stddev"}).collect()[0][0], # стандартное отклонение

df.agg({column: 'min'}).collect()[0][0], # минимум

df.agg({column: 'max'}).collect()[0][0], # максимум

df.agg({column: 'max'}).collect()[0][0] - df.agg({column: 'min'}).collect()[0][0], # диапазон

df.approxQuantile(column, [0.25], 0.001)[0], # 25-й квантиль

df.approxQuantile(column, [0.75], 0.001)[0], # 50-й квантиль

df.approxQuantile(column, [0.95], 0.001)[0] # 75-й квантиль

]

statistics[f'{column}'] = column\_values

Для вычисления статистических показателей, была создана функция, в которую передается фрейм и колонка, которая будет проанализирована. Далее создается словарь с ключом 'Statistic' включающая в себя значения названий метрик, а после с помощью выполняется агрегация каждого статистического показателя с помощью функций agg, approxQuantile, а мода вычисляется с помощью фильтрации по колонке, которая определяет самое часто встречающиеся значения, формирует список по убыванию и берет первый элемент. Полученные статистики для каждого столбца добавляются в словарь statistics.

Мной были обработаны всё выбранные количественные признаки датасета и представлены на рис. 8.

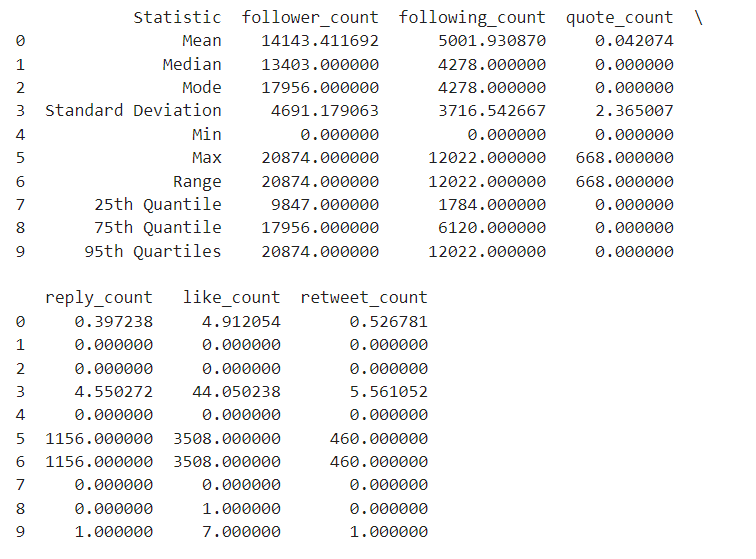


Рисунок 8. Статистические показатели количественных признаков

Благодаря полученным статистическим данным, можно понять какие признаки имеет значение систематизировать и обработать с целью выявления характера и структуры взаимосвязей для получения практических выводов. Из полученных данных можно сделать вывод, что колонки, отвечающие за количество ответов, цитирований, лайков и ретвитов в частности имеют нулевые значения, т.к. мода, медиана, квантили и квартили указывают на это. О том, что во всех колонках есть нулевые значения. Средние значения колонок, указанных ранее, кроме лайков, находятся на уровне нуля.

Далее рассмотрим статистические показатели категориальных признаков.

Категориальные признаки расчитываются с использованием количественных признаков. В данном случае расчитываются:

* 1. Количество (count) - Указывает какое количество значений имеется в колонке.
  2. Процентиль (percentile) - Процентиль это числовое значение, которое делит упорядоченный набор данных на 100 равных частей. Каждый процентиль представляет собой точку данных, ниже которой падает определенная процентная доля значений в наборе данных (в нашем случае: 25%, 50%, 75% и 95%).
  3. Минимум (min) - Минимальный количественный показатель категориального признака.
  4. Максимум (max) - Максимальный количественный показатель категориального признака.
  5. Среднее (Mean) - Среднее значение количественных показателей категориального признака.
  6. Стандартное отклонение (stddev) - Мера разброса данных относительно среднего значения количественных показателей категориального признака.

language\_followers\_df = df.select(df.follower\_count).filter(df.tweet\_language == tweet\_language)

stats = language\_followers\_df.select(

F.percentile\_approx(df.follower\_count, [0.25, 0.5, 0.75, 0.95], 1000000).alias("percentiles"),

F.min(df.follower\_count).alias('min'),

F.max(df.follower\_count).alias('max'),

F.mean(df.follower\_count).alias('mean'),

F.stddev(df.follower\_count).alias('stddev')

).collect()[0]

Разберем, как формируются статистические показатели. Как видно из кода, количественный признак follower\_count фильтруется по категориальному признаку tweet\_language. Формирование статистических показателей происходит с использованием библиотеки «F». Она является функциональной библиотекой pyspark и подключается с помощью строки

import pyspark.sql.functions as F

После формирование статистики, создается локальный список и берется первый элемент, который в дальнейшем визуализируется в виде таблицы. Все виды языков рассматривать не будем, и сократим список до 10 уникальных языков. Результат представлен на рис. 9.

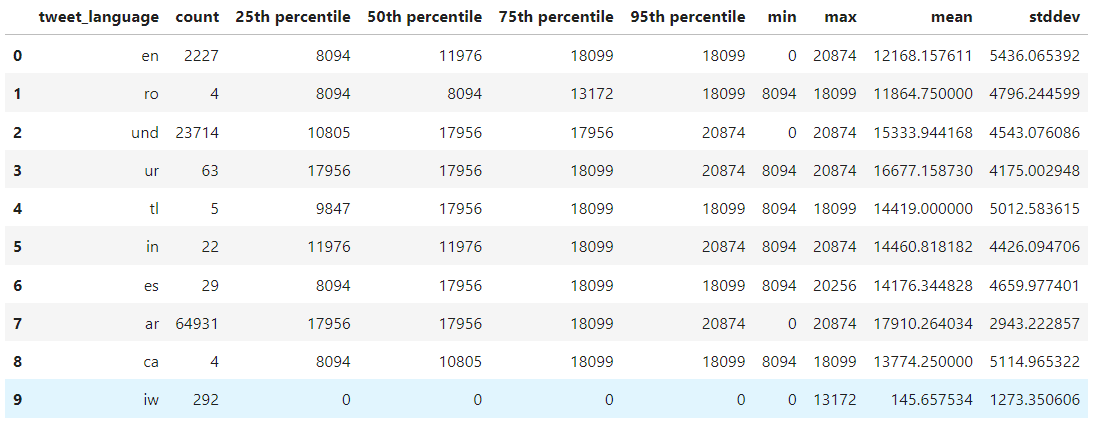


Рисунок 9. Статистические показатели категориального признака колонки «tweet\_language»

Данный статистический анализ позволяет определить, различные количественные показатели сгруппированных данных по категориальному признаку. Количество встречаемых значений данного категориального признака, минимальные и максимальные значения количественных признаков и так далее.

Осталось рассмотреть статистические показатели бинарного признака. См. табл. 2.

Бинарный признак:

* 1. Количество (count) - Указывает какое количество значений имеется в колонке
  2. Проценты (percentage) - Отношение количества значений в процентах данного показателя от обшего.

true\_false\_counts = df.groupBy("is\_retweet").count()

total\_count = df.count()

true\_false\_counts\_with\_percentage = true\_false\_counts.withColumn("percentage", F.col("count") / total\_count \* 100)

true\_false\_counts\_with\_percentage.show()

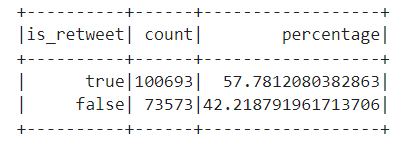


Рисунок 10. Статистические показатели бинарного признака колонки «is\_retweet»

В бинарном признаке ведётся подсчёт количества значений true и false, а на основе полученных значений от общего числа формируется процент от 100. Результаты статистического показателя бинарного признака можно увидеть на рис. 10.

* 1. **Визуализацией распределения наиболее важных признаков**

Для визуализации распределения наиболее важных признаков были использованы следующие графики:

* Гистограммы, в количестве 6 графиков, рис. 11;
* Круговая диаграмма, в количестве 1 графика, рис. 12;
* Диаграмма рассеивания, в количестве 5 графиков, рис 13-17.

Это помогает наглядно увидеть распределение данных и выявить возможные закономерности.

Гистограммы представляют количественные параметры важных признаков количественного типа. Было отмечено 6 колонок.

Программный код для формирования графиков представлен ниже. Все графики колонок формируются с помощью цикла.

columns = ['follower\_count', 'following\_count', 'reply\_count', 'quote\_count', 'like\_count', 'retweet\_count']

fig, axs = plt.subplots(2, 3, figsize=(20, 10))

axs = axs.ravel() # Преобразуем матрицу осей в одномерный массив

for i, column in enumerate(columns):

total\_counts = df.groupBy(column).count().collect()

values = [int(row[column]) for row in total\_counts]

counts = [int(row['count']) for row in total\_counts]

axs[i].bar(values[:100], counts[:100], color='skyblue', edgecolor='red')

axs[i].set\_title(f'Гистограмма распределения {column}')

axs[i].set\_xlabel(f'Значения {column}')

axs[i].set\_ylabel('Общее количество')

plt.tight\_layout()

plt.show()

В данном коде создаем список колонок по которым будем итерироваться. Создаем сетки для графиков. Преобразуем матрицу осей в одномерный массив для удобства итераций по столбцам. После перебираем каждый столбец из списка и для каждого столбца выполняет операцию группировки данных по этому столбцу и подсчета количества. И в конце отображаем диаграмму по первыми 100 значениями и их количеству, чтобы избежать слишком больших графиков. Все графики отображены на одном рис. 11.

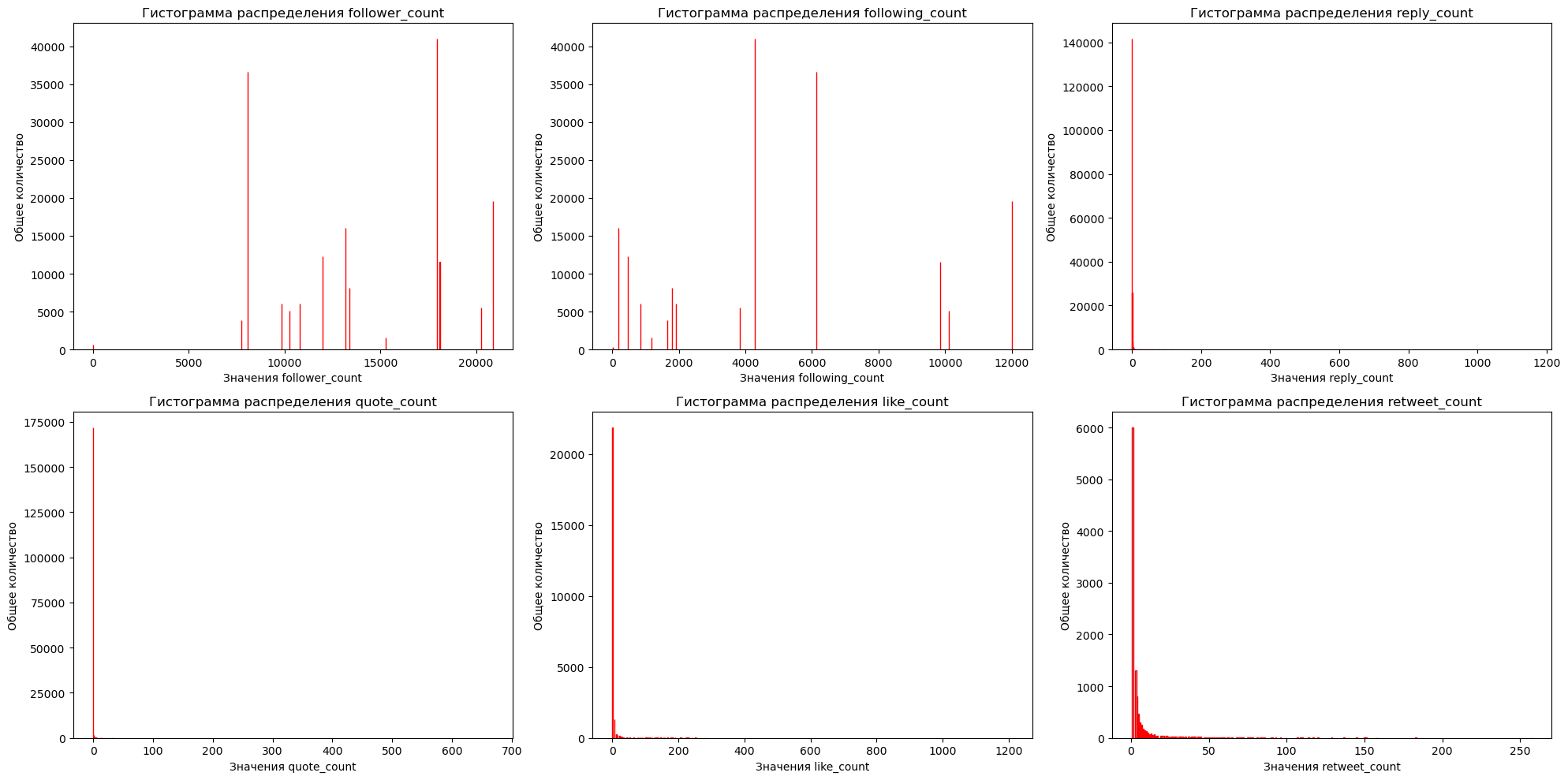


Рисунок 11. Гистограммы распределений количественных признаков на весь датасет

Из представленных гистограмм видно то, что колонки цитирований, ответов, лайков и ретвитов очень близки к одинаковому значению и похожи друг на друга. Их количество в большей степени равны нулю и есть немного строк, в которых имеются метрики. Распределение колонок подписок и подписчиков слабо связаны и разнятся друг с другом.

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

retweet\_counts = df.groupBy("is\_retweet").count().collect()

labels = [str(row['is\_retweet']) for row in retweet\_counts]

sizes = [int(row['count']) for row in retweet\_counts]

colors = ['lightcoral', 'lightskyblue']

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)

plt.axis('equal')

plt.title('Является ли пост ретвитом:', fontsize=14)

plt.show()

Данный код создает круговую диаграмму для визуализации распределения бинарного признака.

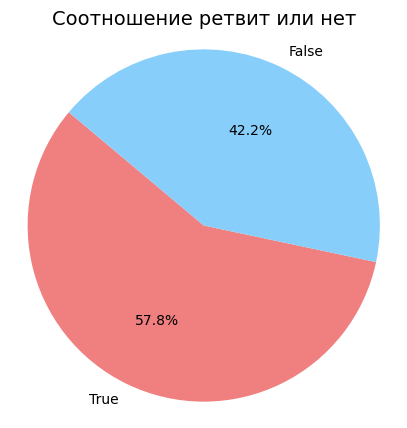


Рисунок 12. Круговая диаграмма бинарного признака

Круговая диаграмма представляют количественные параметры бинарного признака для визуализации данного признака. На рис. 12 представлена диаграмма данного признака.

Программный код для формирования диаграмм рассеивания представлен ниже по одной диаграмме, другие формируются с внесением изменений названий других колонок.

follower\_counts = df.select('follower\_count').collect()

following\_counts = df.select('following\_count').collect()

reply\_counts = df.select('reply\_count').collect()

quote\_counts = df.select('quote\_count').collect()

like\_counts = df.select('like\_count').collect()

retweet\_counts = df.select('retweet\_count').collect()

# Построение диаграммы рассеяния

plt.figure(figsize=(11, 7))

plt.scatter([x['like\_count'] for x in like\_counts], [x['retweet\_count'] for x in retweet\_counts], alpha=0.3, color='skyblue')

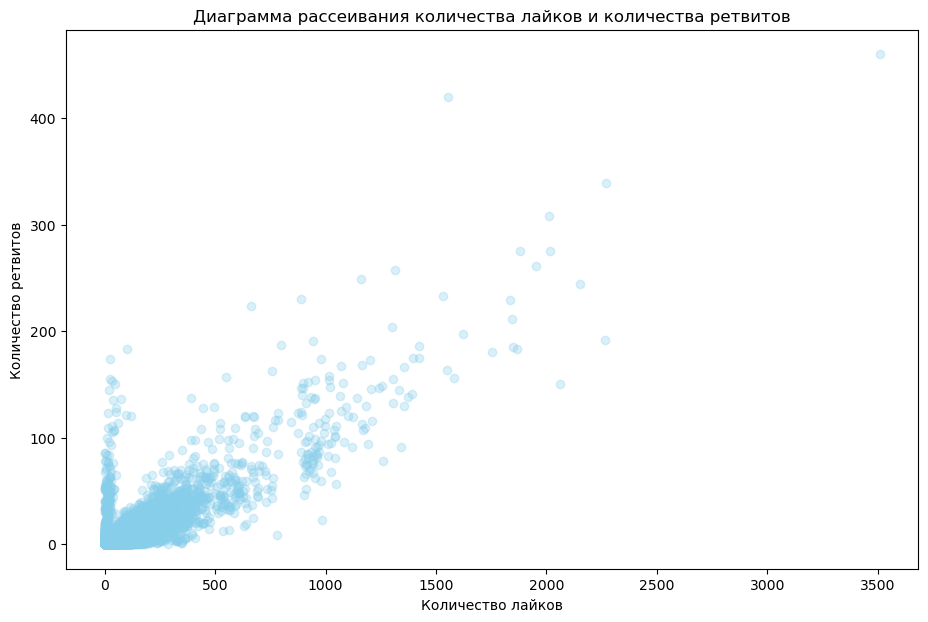
plt.title('Диаграмма рассеивания количества лайков и количества ретвитов')

plt.xlabel('Количество лайков')

plt.ylabel('Количество ретвитов')

plt.show()

Формируем локальные списки по каждой из колонок, по которым будут формироваться диаграммы. А после уже и формируем данные диаграммы, где мы с помощью циклов перебираем все элементы списка определенных колонок.

 Рисунок 13. Диаграмма рассеивания лайков-ретвитов

На рис. 13 видно то, что данные показатели в большинстве находятся у нулевых значений и также в больших случаях могут пересекаться при других метриках, отличных от нуля.

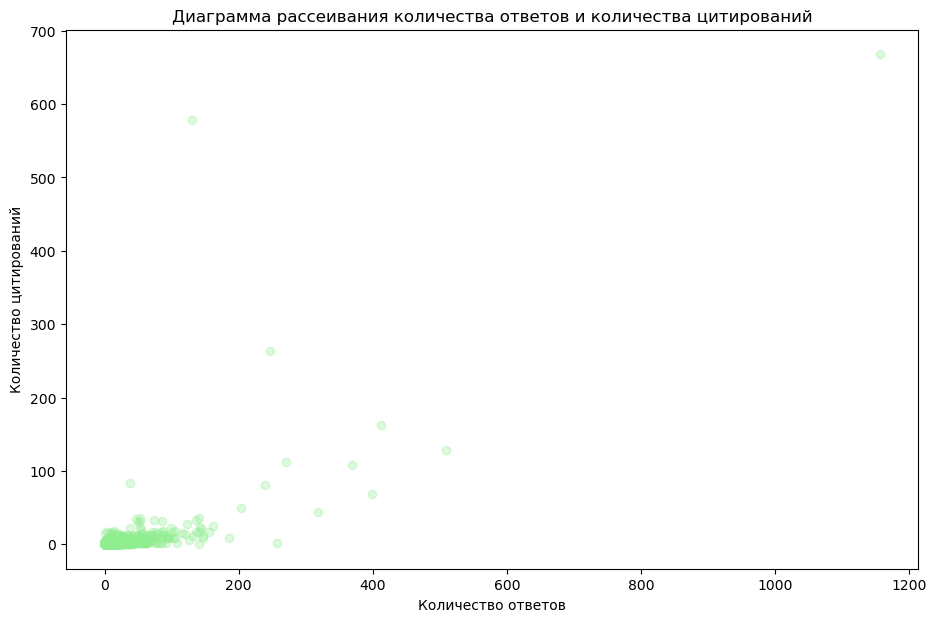


Рисунок 14. Диаграмма рассеивания ответов-цитирований

Здесь (рис. 14) уже рассматриваем две других колонки данные, которых на гистограммах были схожи, а именно количество цитирований и ответов. Видно, что также чаще всего они находятся в нулевом диапазоне и гораздо реже пересекаются в других местах, как на рис. 13.

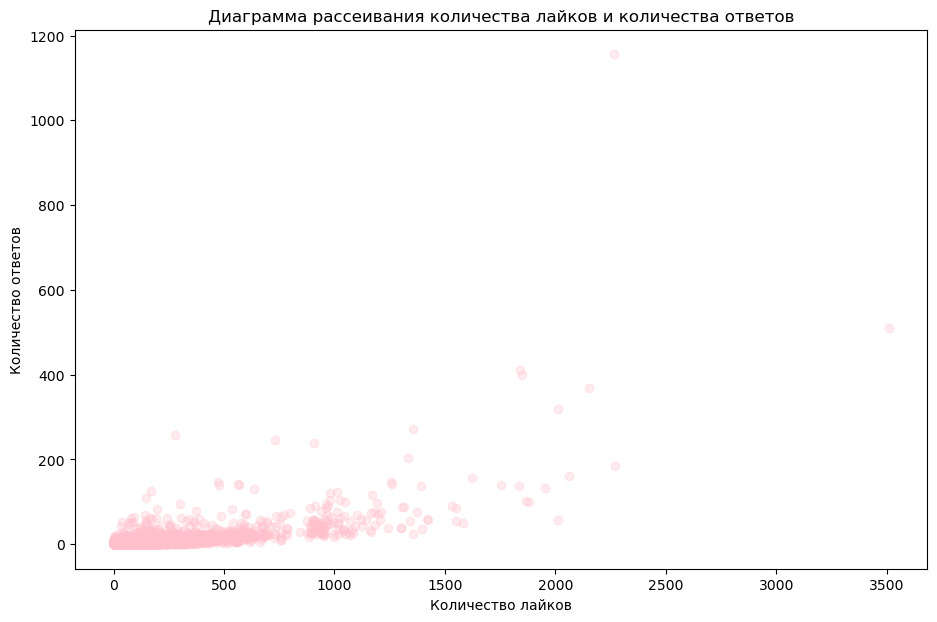


Рисунок 15. Диаграмма рассеивания лайков-ответов

Сравнение количества лайков и количества ответов, с рис. 14, т.к. из предыдущего графика видно, что количество ответов может встречаться чаще в большей метрике, чем цитирований. И на рис. 15 видно, что множество точек пересекаются от диапозона 0-1000 с постепенным увеличением ответов.

Такие графики можно сформировать для всех 4 колонок, которые очень близки по значению друг с другом и получится 6 различных графиков. Но у нас еще есть поля подписчиков и подписок.

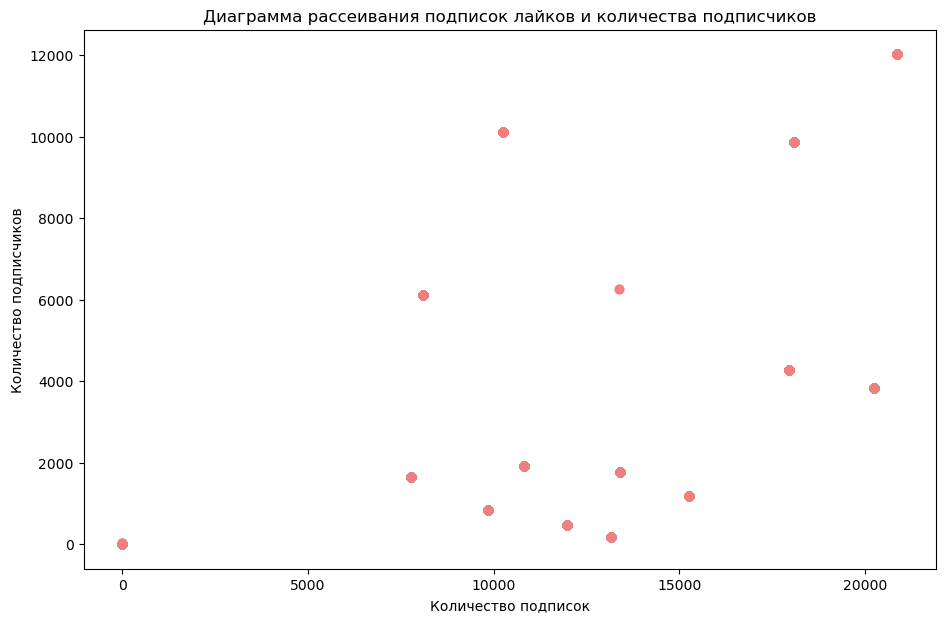


Рисунок 16. Диаграмма рассеивания подписчики-подписок

Из графика (рис. 16) видно, что колонки никак не связаны и их значения просто разбросаны по диаграмме. Из диаграммы видно, что из-за того, что точек мало, значения этих показателей часто совпадают друг с другом. Это говорит о том, что возможно это могут быть уникальные пользователи, если не брать в расчёт точку с двумя нулями. Ведь редко когда у уникальных пользователей могут совпасть метрики.

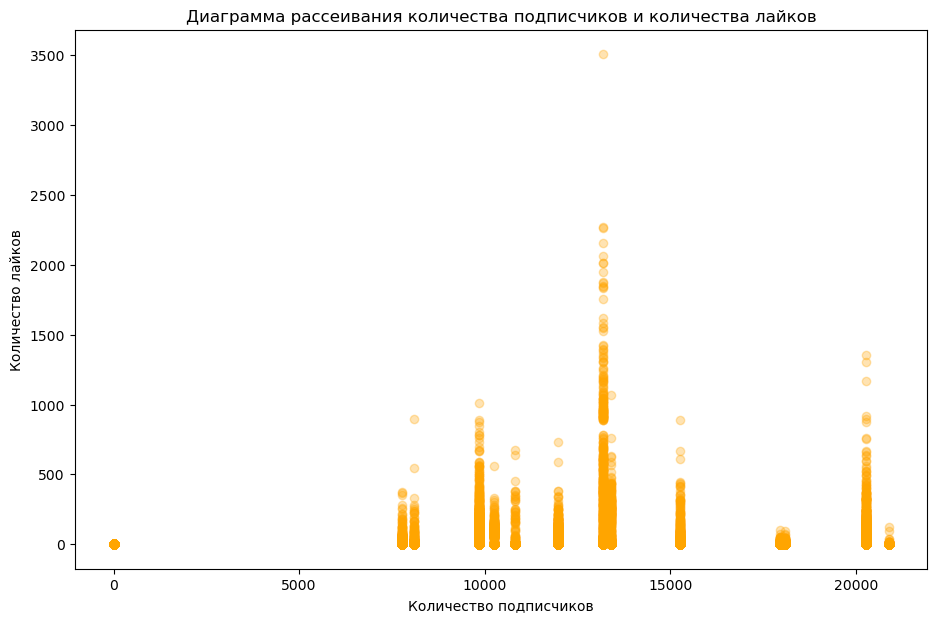


Рисунок 17. Диаграмма рассеивания подписчиков-лайков

Из графика (рис. 17) видно, что при любом количестве подписчиков у пользователя и при увеличении метрики лайков точки становятся все прозрачней. Это говорит о том, что множество постов остаются с малым количеством активности вне зависимости от популярности пользователя.

* 1. **Корреляций между признаками**

Чтобы выявить связи между признаками, нужно построить матрицу корреляций.

Поля "tweetid" и "userid" нет необходимости включать в списки для анализа корреляции. Идентификаторы обычно не имеют значения для анализа корреляций и могут даже помешать получению точных результатов.

Корреляцию будем считать только по количественным признакам. В данном датасете их шесть:

1. 'follower\_count' - Количество подписчиков

2. 'following\_count' - Количество подписок

3. 'quote\_count' - Количество цитирований

4. 'reply\_count' - Количество ответов

5. 'like\_count' - Количество лайков

6. 'retweet\_count' - Количество ретвитов

columns\_to\_exclude = ['tweetid', 'userid', 'tweet\_language', 'is\_retweet']

list\_columns = [column for column in df.columns if column not in columns\_to\_exclude]

assembler = VectorAssembler(inputCols=list\_columns, outputCol="features")

assembler.setHandleInvalid("skip")

df\_vector = assembler.transform(df).select("features")

Данный код исключает колонки, подсчёт которых не требуется. Формируем список из оставшихся колонок. С помощью VectorAssembler создаем вектор непрерывных числовых признаков, где пропущенные значения пропускаются и после формируем датафрейм.

matrix = Correlation.corr(df\_vector, "features").collect()[0][0]

corrmatrix = matrix.toArray().tolist()

columns = ['follower\_count', 'following\_count', 'quote\_count', 'reply\_count', 'like\_count', 'retweet\_count']

df\_corr = spark.createDataFrame(corrmatrix, columns)

df\_corr.select(

round('follower\_count', 3).alias('follower\_count'),

round('following\_count', 3).alias('following\_count'),

round('quote\_count', 3).alias('quote\_count'),

round('reply\_count', 3).alias('reply\_count'),

round('like\_count', 3).alias('like\_count'),

round('retweet\_count', 3).alias('retweet\_count')

).show()

Формируем матрицу корреляции, затем преобразуем полученную матрицу в массив, а затем в список. Создаем список с именами столбцов, для которых мы хотим отобразить корреляционную матрицу. Затем создаем новый датафрейм по матрице и колонкам и в конце выполняем выборку столбцов и округляем значения до трех знаков после запятой.

Матрица корреляции выглядит следующим образом. См. Табл. 2.

Таблица 2. Матрица корреляции

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| follower\_count | following\_count | quote\_count | reply\_count | like\_count | retweet\_count |
| 1.0 | 0.38 | 0.001 | -0.016 | -0.02 | 0.01 |
| 0.38 | 1.0 | -0.015 | -0.063 | -0.096 | -0.057 |
| 0.001 | -0.015 | 1.0 | 0.707 | 0.302 | 0.241 |
| -0.016 | -0.063 | 0.707 | 1.0 | 0.654 | 0.544 |
| -0.02 | -0.096 | 0.302 | 0.654 | 1.0 | 0.857 |
| 0.01 | -0.057 | 0.241 | 0.544 | 0.857 | 1.0 |

Обнаружено множество отрицательных корреляций между числом подписчиков/подписок и ответами/цитирований/лайками/ретвитами.

У колонок "reply\_count", "like\_count", "retweet\_count" и "quote\_count" корреляция находятся в диапозон 0.241 – 0.857.

Самая сильная связь имеется у колонок "like\_count" и "retweet\_count".

Корреляции "диагональных" признаков всегда равны 1, т.к. это и есть те же самые признаки и они всегда будут изменяться, если их изменяют.

* 1. **Выводы**

В ходе работы был изучен функционал Apache PySpark. Я научился пользоваться функционалом Apache Spark благодаря Python API - PySpark. Целью работы было провести разведочный анализ датасета. В ходе работы были выполнены данные пункты:

* Типы признаков в датасете: Определил, какие данные включены в датасет (числовые, категориальные и бинарные).
  + Пропущенные значения и их устранения: Определил наличие пропущенных значений в данных и их устранил. Строки с пропущенными значениями было решено полностью удалить.
  + Выбросы и их устранения: Провел работу по устранению выбросов используя метод межквартильного интервала, а после проверил их отсутствие с помощью boxspot-ов.
  + Расчет статистических показателей признаков (средних, квартилей и т.д.): Провел расчёт статистических показателей категориальных, бинарных и числовых признаков, которые включают: средние значения, медианы, квартили, моды, стандартные отклонения, диапазоны значений, квантили, процентные соотношения и другие характеристики, чтобы лучше понять распределение данных.
  + Визуализация распределения наиболее важных признаков: Выполнил визуализацию распределения наиболее важных признаков. Визуализация помогает визуально оценить данные и выявить особенности.
  + Корреляция между признаками: Исследовал корреляции между признаками, позволяющие понять, какие признаки взаимосвязаны между собой, а какие – нет.

Разведочный анализ помогает лучше понять характеристики датасета, подготовить данные для построения моделей и выявить интересные закономерности, которые могут стать основой для более глубокого исследования.

1. **Машинное обучение на больших данных**
   1. **Постановка задачи**

Цель и задачи работы:

1. Познакомиться с базовыми алгоритмами машинного обучения;

2. Познакомиться с реализацией машинного обучения в библиотеке Spark ML.

3. Получить навыки разработки программного обеспечения для анализа данных с использованием pyspark.

Необходимо выполнить анализ обработанного датасета с помощью двух алгоритмов машинного обучения:

* Задача регрессии - GradientBoostingMachine
* Задача бинарной классификации - LogisticRegression

При анализе датасета предпочтительно использовать признаки, показавшие наилучшую корреляцию при выполнении разведочного анализа. Для задачи классификации использовать бинарный признак.

Необходимо выполнить обучение и валидацию модели, рассчитайте значения метрик классификации и регрессии. Выполните подбор гиперпараметров моделей по сетке.

* 1. **Применение алгоритмов подготовки данных и машинного обучения**

Необходимо подключить библиотеки для обучения модели.

После осуществления разведочного анализа был сохранен датасет. Подключаем все необходимые библиотеки и отфильтрованный датасет. Список настроек представлен в виде программного кода ниже.

from pyspark.sql.types import \*

from pyspark.sql.functions import \*

from pyspark.sql import SparkSession

from pyspark.ml import Pipeline

from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer, MinMaxScaler

from pyspark.ml.regression import GBTRegressor

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

from pyspark.ml.evaluation import RegressionEvaluator, BinaryClassificationEvaluator

spark = SparkSession.builder.master("local[\*]").getOrCreate()

filename\_data = 'processed\_data/iran\_201906\_2\_tweets\_csv\_hashed.csv'

csv = spark.read.csv(filename\_data, inferSchema=True, header=True)

csv.show(10)

csv.printSchema()

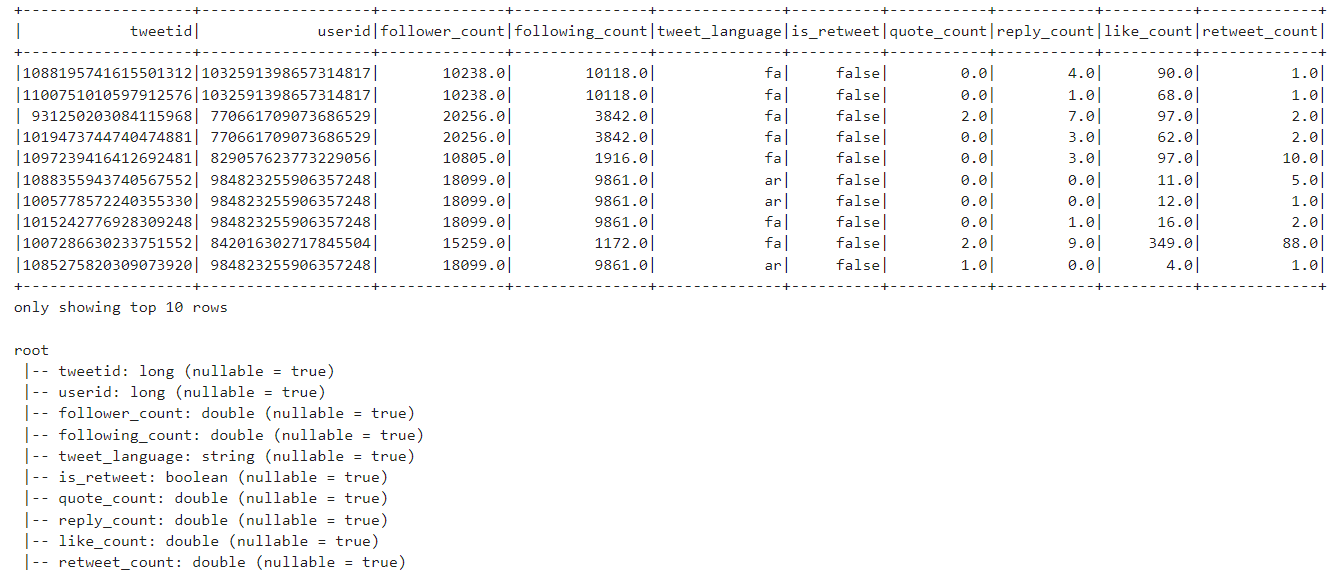


Рисунок 18. Результат вывода значений из обработанного датасета

Подключаем библиотеки для решений задач регрессии и классификации, для проведения кросс-валидации и формирования сетки гиперпараметров, библиотеки для обучения моделей и так далее.

И выведем первые 10 строк датасета и его типы колонок для проверки подключения, результат вывода изображен на рисунке 18.

* 1. **Задача регрессии – GradientBoostingMachine**

Для начала необходимо отсортировать колонки и выбрать только те, что показали наилучшую корреляцию в разведочном анализе. Было решено использовать такие колонки, как:

1. 'quote\_count' - Количество цитирований

2. 'reply\_count' - Количество ответов

3. 'like\_count' - Количество лайков

4. 'retweet\_count' - Количество ретвитов

data = csv.select("quote\_count", "reply\_count", "like\_count", "retweet\_count")

Сохраним эти колонки в переменной data.

splits = data.randomSplit([0.7, 0.3])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, " Testing Rows:", test\_rows)

Далее разделим данные на обучающую и тестовую выборку: Будем использовать 70% данных для обучения и 30% для тестирования.

После применения алгоритмы имеем на выходе: Тренировочных строк – 121974 и тестовых строк – 52292.

Осуществим разделение датасета и приступаем к определению конвейера, настройки конвейера:

Конвейер состоит из серии этапов преобразования и оценки, которые обычно подготавливают фрейм данных для моделирования, а затем обучают прогнозирующую модель. В этом случае вы создадите конвейер с двумя этапами:

1. VectorAssembler: Создает вектор непрерывных числовых признаков.

2. Gradient-boosted tree regression: Обучает модель регрессии.

numVect = VectorAssembler(inputCols = ["quote\_count", "reply\_count", "retweet\_count"], outputCol="features")

gbtr = GBTRegressor(featuresCol = 'features', labelCol='like\_count', maxIter = 5, maxDepth=2)

pipeline = Pipeline(stages=[numVect, gbtr])

Как видно, будем предсказывать значение колонки "like\_count" на основе данных из колонок "quote\_count", "reply\_count" и "retweet\_count" с применением в задачи регрессии 5 итераций градиентного бустинга и с глубиной дерева равной 2.

После настройки конвейера, запускаем его на обучаемых данных с помощью команды: piplineModel = pipeline.fit(train).

А после этого, на основе тестовых данных, генерируем предсказываемые значения, отображенных на рис.19.

prediction = piplineModel.transform(test)

predicted = prediction.select("features", "prediction", "like\_count")

predicted.show(20)

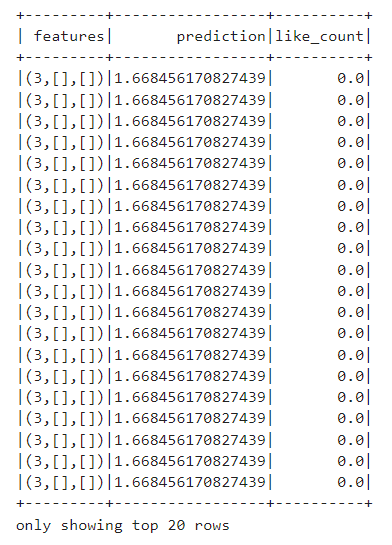


Рисунок 19. Предсказания на основе тестовых данных в задачи регрессии

regressionEvaluator = RegressionEvaluator(predictionCol="prediction", labelCol="like\_count", metricName="rmse")

# RMSE

rmse = regressionEvaluator.evaluate(prediction)

print(f"The RMSE for the Gradient-boosted tree regression model is {rmse:0.2f}")

# MSE

mse = regressionEvaluator.setMetricName("mse").evaluate(prediction)

print(f"The MSE for the Gradient-boosted tree regression model is {mse:0.2f}")

# R2

r2 = regressionEvaluator.setMetricName("r2").evaluate(prediction)

print(f"The R2 for the Gradient-boosted tree regression model is {r2:0.2f}")

# MAE

mae = regressionEvaluator.setMetricName("mae").evaluate(prediction)

print(f"The MAE for the Gradient-boosted tree regression model is {mae:0.2f}")

А для оценки модели используем такие метрики, как:

1. RMSE (Root Mean Square Error) - Это классическая метрика, используемая для измерения точности модели в задачах регрессии. RMSE представляет собой квадратный корень из среднеквадратичной ошибки (MSE) и позволяет оценить, насколько сильно отличаются предсказанные значения от фактических. Чем ближе RMSE к нулю, тем лучше модель справляется с прогнозированием. RMSE подвержен влиянию на выбросы в данных, так как он учитывает квадраты отклонений предсказаний от фактических значений.

2. MSE (Mean Squared Error) - Является средним значением квадратов разностей между предсказанными и фактическими значениями. Чем ниже значение MSE, тем лучше качество модели.

3. R2 (R-squared) - Это коэффициент детерминации, который предоставляет информацию о том, как хорошо модель соотносится с данными. R2 принимает значения от 0 до 1, где 1 означает идеальное соответствие модели данным. Он позволяет сравнивать модель с простым средним значением зависимой переменной. Чем ближе R2 к 1, тем лучше модель объясняет изменчивость зависимой переменной.

4. MAE (Mean Absolute Error) - Это среднее абсолютное значение разностей между предсказанными и фактическими значениями. MAE менее подвержен выбросам, чем MSE и RMSE, так как не возводит отклонения в квадрат и, следовательно, не учитывает их взаимную агрессию.

В результате предсказаний имеем такие результаты метрик, как в табл.3:

Таблица 3. Метрики модели регрессии

|  |  |
| --- | --- |
| Метрика: | Значение: |
| RMSE: | 21.04 |
| MSE: | 442.65 |
| R2: | 0.76 |
| MAE: | 4.08 |

Полученные данные можно оптимизировать и улучшить показатели метрик с помощью применения параметрической сетки и кросс-валидации модели.

param\_grid = (ParamGridBuilder()

.addGrid(gbtr.maxDepth, [2, 4, 6])

.addGrid(gbtr.maxBins, [20, 60])

.addGrid(gbtr.maxIter, [10, 20])

.build())

Установим такие гиперпараметры модели градиентного бустинга:

* Задаем значения для максимальной глубины деревьев: 2, 4 и 6.
* Устанавливаем различные значения для максимального числа бинов: 20 и 60.
* Определяем различные значения для максимального числа итераций: 10 и 20.

cv = CrossValidator(estimator=pipeline, \

estimatorParamMaps=param\_grid, \

evaluator=RegressionEvaluator(

predictionCol="prediction", \

labelCol="like\_count", \

metricName="rmse"), \

numFolds=2)

И передаем эти параметры в метод кросс-валидации. Запускаем конвейер на обучаемых данных (cv\_model = cv.fit(train)) и генерируем предсказания на тестовых (newPrediction = cv\_model.transform(test)), как и ранее. На этот раз у нас имеется больше параметров и в результате оценщика оптимальных параметров имеем такие результаты метрик, указанных в таблице 4:

Таблица 4. Метрики модели регрессии после кросс-валидации

|  |  |
| --- | --- |
| Метрика: | Значение: |
| RMSE: | 15.37 |
| MSE: | 236.22 |
| R2: | 0.87 |
| MAE: | 3.02 |

* 1. **Задача классификации – LogisticRegression**

Для задачи классификации будет использован бинарный признак «is\_retweet», который уже был в датасете изначально, поэтому создавать новый признак не имеет смысла.

csv = csv.withColumn("label", when(col("is\_retweet") == True, 1).otherwise(0))

csv.show(10)

csv.printSchema()

Для начала необходимо преобразовать колонку, в которой будут значения: 1 – true / 0 – false. Данную колонку решено было назвать «label». По итогу сформированный датасет для создания модели влючает следующие колонки, изображение данных можно увидеть на рис. 20:

1. 'follower\_count' - Количество подписчиков

2. 'following\_count' - Количество подписок

3. 'quote\_count' - Количество цитирований

4. 'reply\_count' - Количество ответов

5. 'like\_count' - Количество лайков

6. 'retweet\_count' - Количество ретвитов

7. 'label' – Бинарный признак является ли пост ретвитом

data = csv.select("follower\_count", "following\_count", "quote\_count", "reply\_count", "like\_count", "retweet\_count", "label")

data.show(20)

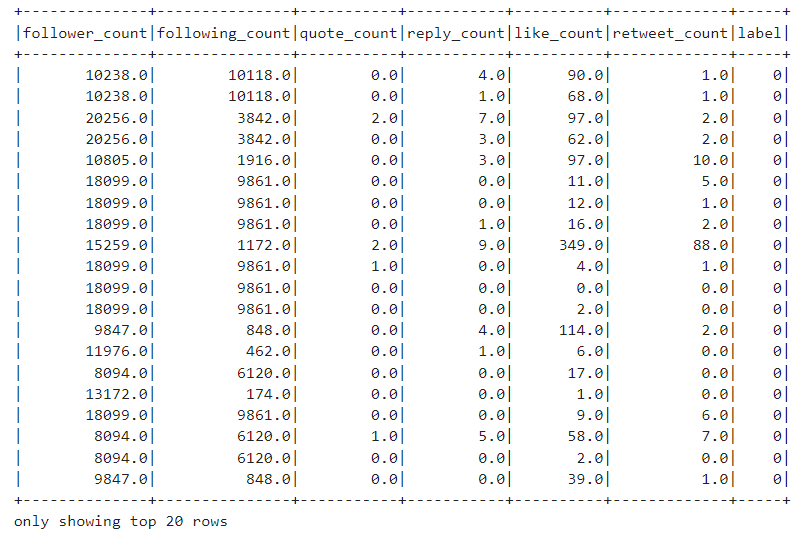


Рисунок 20. Признаки, использованные в задачи классификации

Также определим количество значений 0 и 1 в колонке «label» (рис. 21).

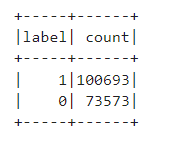


Рисунок 21. Количество значений к колонке «label».

Из количества полученных значений видно, что датасет сбалансирован, т.к. отношение 1/0 ~ 57%/43%.

splits = data.randomSplit([0.7, 0.3])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, " Testing Rows:", test\_rows)

Повторно формируем тренировочные и тестовые выборки, с такими же процентами, где тренировочных строк – 121888, а тестовых – 52378.

Осуществим разделение датасета приступаем к определению конвейера:

Конвейер состоит из серии этапов преобразования и оценки, которые обычно подготавливают фрейм данных для моделирования, а затем обучают прогнозирующую модель. В этом случае мы также создаем конвейер с тремя этапами:

1. VectorAssembler: Создает вектор непрерывных числовых признаков.

2. MinMaxScaler: Нормализует непрерывные числовые характеристики.

3. Logistic Regression: Обучает модель классификации.

numVect = VectorAssembler(inputCols = ["follower\_count", "following\_count", "quote\_count", "reply\_count", "like\_count", "retweet\_count"], outputCol="numFeatures")

minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol="features")

lr = LogisticRegression(labelCol="label",

featuresCol="features",

maxIter=2,

regParam=0.3)

pipeline = Pipeline(stages=[numVect, minMax, lr])

Как видно, будем предсказывать значение колонки "label" на основе данных из колонок "quote\_count", "reply\_count", "retweet\_count", "like\_count", "following\_count" и "follower\_count" с применением в задачи классификаций 2 итераций логистической регрессии и параметр регуляризации равным 0.3, который помогает предотвратить переобучение путем штрафования больших весов в модели.

После настройки конвейера, запускаем его на обучающих данных (piplineModel = pipeline.fit(train)).

А после этого, на основе тестовых данных, генерируем предсказываемые значения (рис.22):

prediction = piplineModel.transform(test)

predicted = prediction.select("features", "prediction", "label")

predicted.show(20)

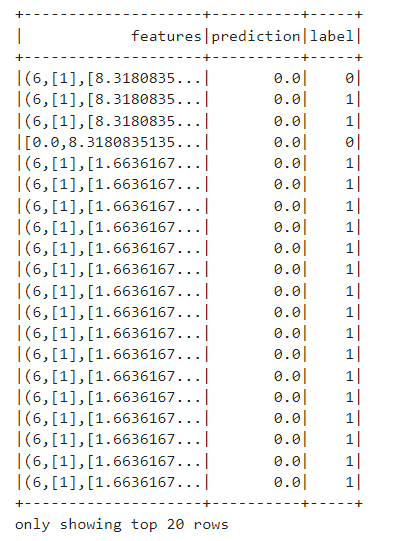


Рисунок 22. Предсказания на основе тестовых данных в задачи классификации

tp = float(predicted.filter("prediction == 1.0 AND label == 1").count())

fp = float(predicted.filter("prediction == 1.0 AND label == 0").count())

tn = float(predicted.filter("prediction == 0.0 AND label == 0").count())

fn = float(predicted.filter("prediction == 0.0 AND label == 1").count())

pr = tp / (tp + fp)

re = tp / (tp + fn)

metrics = spark.createDataFrame([

("TP", tp),

("FP", fp),

("TN", tn),

("FN", fn),

("Precision", pr),

("Recall", re),

("F1", 2\*pr\*re/(re+pr))],["metric", "value"])

metrics.show()

А для оценки модели классификации используем такие метрики, как:

1. Первая метрика это True Positive (TP). TP представляет собой количество истинных положительных результатов, т.е. случаев, когда модель правильно предсказала положительный результат.
2. Далее у нас есть False Positive (FP), который отражает количество ложноположительных результатов. Это случаи, когда модель неправильно предсказала положительный результат.
3. True Negative (TN), который показывает количество истинно отрицательных результатов - ситуации, когда модель правильно предсказала отрицательный результат.
4. И False Negative (FN) - количество ложноотрицательных результатов, т.е. случаи, когда модель неправильно предсказала отрицательный результат.
5. Precision (точность) трактуется как отношение TP к сумме TP и FP. Это означает, что точность измеряет, насколько много из предсказанных моделью положительных случаев действительно являются положительными.
6. Recall (полнота) определяется как отношение TP к сумме TP и FN. Полнота оценивает, насколько много истинных положительных случаев было правильно предсказано моделью.
7. Наконец, F1-мера - это гармоническое среднее между точностью и полнотой. Она вычисляется как 2 \* Precision \* Recall / (Precision + Recall) и предоставляет баланс между точностью и полнотой модели.

На выходе, после обучения модели, имеем метрики, указанные в табл. 5.

Таблица 5. Метрики модели классификации

|  |  |
| --- | --- |
| Metric | value |
| TP | 29000.0 |
| FP | 10507.0 |
| TN | 11547.0 |
| FN | 1324.0 |
| Precision | 0.7340471308881971 |
| Recall | 0.9563382139559425 |
| F1 | 0.8305766779796939 |

Также есть возможность осуществить оценку качества модели бинарной классификации с использованием метрики AUR.

evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

aur = evaluator.evaluate(prediction)

print ("AUR = ", aur)

AUC (Area Under the ROC Curve) - это метрика, используемая для оценки качества модели бинарной классификации. Она представляет собой площадь под кривой ROC (Receiver Operating Characteristic).

Кривая ROC строится путем варьирования порога классификации и вычисления двух показателей: True Positive и False Positive.

AUC-ROC описывает производительность модели на всем спектре возможных порогов классификации, объединяя в себе информацию из всех возможных значений порога. Чем ближе значение AUC к 1, тем лучше модель способна различать между классами. Если AUC = 0.5, это означает, что модель дает случайные предсказания без различия между классами.

Метрика AUC, в обученной модели, равна: 0.8118560037074648

paramGrid = ParamGridBuilder().addGrid(lr.regParam, np.arange(0, 1, 0.1)).addGrid(lr.maxIter, [10, 20]).build()

cv = CrossValidator(estimator=pipeline, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid,

numFolds=2)

Для метода бинарной классификации также можно применять гиперпараметры модели по сетке и кросс-валидацию. В нашем случае, параметры сетки для модели: варьируемые значения параметра регуляризации [0.1, 1.0] с шагом в 0.1 и параметры максимального числа итераций равные 10 и 20, а также разделений количества фолдов равному 2.

И с помощью кросс-валидации осуществим поиск наилучшего значения. Запускаем конвейер(cv\_model = cv.fit(train)), генерируем предсказания на тестовых данных (newPrediction = cv\_model.transform(test)), а после формируем метрики, которые указаны в табл. 6:

Таблица 6. Метрики модели классификации после кросс-валидации

|  |  |
| --- | --- |
| Metric | value |
| TP | 30324.0 |
| FP | 3686.0 |
| TN | 18368.0 |
| FN | 0.0 |
| Precision | 0.8916201117318435 |
| Recall | 1.0 |
| F1 | 0.9427052569403426 |

А также метрика AUC равна: 0.91643239321665

* 1. **Анализ результатов**

В ходе решений задач регрессии и классификации были получены некоторые результаты. В ходе задачи регрессии метрики описаны в табл. 7.

Таблица 7. Анализ метрик задачи регрессии

|  |  |  |
| --- | --- | --- |
|  | До кросс-валидации | После кросс-валидации |
| RMSE: | 21.04 | 15.37 |
| MSE: | 442.65 | 236.22 |
| R2: | 0.76 | 0.87 |
| MAE: | 4.08 | 3.02 |

В ходе анализа полученных результатов из табл. 7, видно то, что метрики RMSE, MSE и MAE уменьшились, а метрика R2 – увеличилась. То есть в результате уменьшения метрики RMSE – модель лучше справляется с прогнозированием, т.к. показатель стремится к 0. MSE – качество модели стала лучше, т.к. эта метрика в идеале также стремится к 0. Из-за MAE – модель также лучше справляется с прогнозированием, т.к. тоже показатель должен уменьшаться. А вот R2 – стал больше, что говорит о изменении коэффициента детерминации и говорит о том, что модель лучше справляется с предсказанием, чем изначально. В итоге имеем, что изменение всех показателей после кросс-валидации улучшили модель предсказания.

Таблица 8. Анализ метрик задачи классификации

|  |  |  |
| --- | --- | --- |
|  | До кросс-валидации | После кросс-валидации |
| TP: | 29000.0 | 30324.0 |
| FP: | 10507.0 | 3686.0 |
| TN: | 11547.0 | 18368.0 |
| FN: | 1324.0 | 0.0 |
| Precision | 0.7340471308881971 | 0.8916201117318435 |
| Recall | 0.9563382139559425 | 1.0 |
| F1 | 0.8305766779796939 | 0.9427052569403426 |
| AUC | 0.8118560037074648 | 0.91643239321665 |

В ходе анализа результатов, указанных в табл. 8, задачи классификации видно то, что с применением кросс-валидации значения показателей сильно изменились. Метрика AUC увеличилась, что говорит о том, что модель стала лучше различать классы, а не ставит случайные значения. Значение метрии TP и TN – увеличились, а FP и FN – уменьшились, т.е. модель стала лучше предсказывать значения. Из-за изменений ранее указанных метрик значенияPrecision, Recall и F1 также изменились, т.к. напрямую зависят от предсказанных значений. Метрика Precision увеличилась из-за того, что TP – увеличилось, а FP – уменьшилось. Метрика Recall стала равна единице, т.к. FN равно нулю. А F1 увеличилось из-за изменений Precision и Recall.

* 1. **Вывод**

В ходе работы был изучен функционал Apache PySpark. Я научился пользоваться функционалом машинного обучения Apache Spark благодаря Python API - PySpark. Целью работы было провести машинное обучение на больших данных. Был выполнен анализ с помощью двух алгоритмов машинного обучения:

Задача регрессии: GradientBoostingMachine

Задача бинарной классификации: LogisticRegression

Было выполнено обучение и валидация модели, рассчитаны значения метрик классификации и регрессии, а также выполнен подбор гиперпараметров моделей по сетке.

**Заключение**

В ходе выполнения курсовой работы был выполнен разведочный анализ данных с помощью PySpark, а также выполнено машинное обучение на больших данных.

Были получены следующие результаты:

* 1. Определены признаки датасета;
  2. Устранены пропущенные значения и выбросы;
  3. Определены статистические показатели и визуализированы с помощью диаграмм распределения признаки;
  4. Выявлены корреляции признаков;
  5. Выполнено обучение и валидация моделей;
  6. Рассчитаны значения метрик классификации и регрессии.

Поставленная цель работы достигнута, и задачи исследования полностью решены. Результаты работы могут быть приняты для пользования в образовательных целях и личного пользования.

Таким образом, мы приходим, к следующему выводу: благодаря использованию PySpark для разведочного анализа данных и машинного обучения с библиотекой ML Pipeline. Благодаря этому получаем возможность обрабатывать и анализировать большие объемы данных, применять алгоритмы для машинного обучения. Это открывает новые горизонты для исследований, анализа и применения машинного обучения в реальном масштабе, улучшая прогностическую способность данных и способствуя принятию обоснованных решений.

# Список используемых источников

1. PySpark Overview – ApacheSpark [электронный ресурс]. – Режим доступа: https://spark.apache.org/docs/latest/api/python/index.html (дата обращения: 14.12.2023).
2. Classification and regression – ApacheSpark 3.5.0 [электронный ресурс]. – Режим доступа: https://spark.apache.org/docs/latest/ml-classification-regression.html (дата обращения: 14.12.2023).
3. Zhiguo Gou, Yizhou Zeng, Analysis of Spatio-temporal Behavior of Research Travelers Based on Spark Big Data Computing Model. 2023, Procedia Computer Science, Volume 228, Pages 494-501. DOI: 10.1016/j.procs.2023.11.056. Ссылка: https://www.sciencedirect.com/science/article/pii/S187705092301880X (дата обращения: 14.12.2023).
4. Tinku Singh, Shivam Gupta, Satakshi, Manish Kumar, Performance Analysis and Deployment of Partitioning Strategies in Apache Spark. 2023, Procedia Computer Science, Volume 218, Pages 594-603. DOI: 10.1016/j.procs.2023.01.041. Ссылка: https://www.sciencedirect.com/science/article/pii/S1877050923000418 (дата обращения: 14.12.2023).
5. Salloum, S., Dautov, R., Chen, X. et al. Big data analytics on Apache Spark. Int J Data Sci Anal 1, 145–164 (2016). https://doi.org/10.1007/s41060-016-0027-9 (дата обращения: 14.12.2023).
6. Brahmane, A.V., Krishna, B.C. Big data classification using deep learning and apache spark architecture. Neural Comput & Applic 33, 15253–15266 (2021). https://doi.org/10.1007/s00521-021-06145-w (дата обращения: 14.12.2023).
7. Mostafaeipour, A., Jahangard Rafsanjani, A., Ahmadi, M. et al. Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. J Supercomput 77, 1273–1300 (2021). https://doi.org/10.1007/s11227-020-03328-5 (дата обращения: 14.12.2023).
8. Guo, Y., Zhang, Z., Jiang, J. et al. Model averaging in distributed machine learning: a case study with Apache Spark. The VLDB Journal 30, 693–712 (2021). https://doi.org/10.1007/s00778-021-00664-7 (дата обращения: 14.12.2023).
9. Aziz, K., Zaidouni, D. & Bellafkih, M. Leveraging resource management for efficient performance of Apache Spark. J Big Data 6, 78 (2019). https://doi.org/10.1186/s40537-019-0240-1 (дата обращения: 14.12.2023).
10. A.N.M. JayaLakshmi, K.V. Krishna Kishore, Performance evaluation of DNN with other machine learning techniques in a cluster using Apache Spark and MLlib. 2022, Journal of King Saud University - Computer and Information Sciences, Volume 34, Issue 1, Pages 1311-1319. DOI: 10.1016/j.jksuci.2018.09.022. Ссылка: https://www.sciencedirect.com/science/article/pii/S131915781830212X (дата обращения: 14.12.2023).

**Приложение А**

MAX\_MEMORY = '8G'

# Initialize a spark session.

conf = pyspark.SparkConf().setMaster("local[\*]") \

.set('spark.executor.heartbeatInterval', 10000) \

.set('spark.network.timeout', 10000) \

.set("spark.core.connection.ack.wait.timeout", "3600") \

.set("spark.executor.memory", MAX\_MEMORY) \

.set("spark.driver.memory", MAX\_MEMORY)

def init\_spark():

spark = SparkSession \

.builder \

.appName("Pyspark guide") \

.config(conf=conf) \

.getOrCreate()

return spark

spark = init\_spark()

filename\_data = 'data/iran\_201906\_2\_tweets\_csv\_hashed.csv'

# Load the main data set into pyspark data frame

df = spark.read.csv(filename\_data,header=True, mode="DROPMALFORMED", inferSchema=True)

from pyspark.sql.functions import col, array

df = df.drop("user\_profile\_description", "account\_creation\_date", "user\_reported\_location", "account\_language", "tweet\_text", "tweet\_time", "tweet\_client\_name", "in\_reply\_to\_userid", "in\_reply\_to\_tweetid", "quoted\_tweet\_tweetid", "retweet\_userid", "retweet\_tweetid", "latitude", "longitude", "user\_mentions", "poll\_choices", "hashtags", "urls", "user\_screen\_name", "user\_display\_name", "user\_profile\_url")

df = df.withColumn("tweetid", col("tweetid").cast(LongType()))

df = df.withColumn("userid", col("userid").cast(LongType()))

df = df.withColumn("follower\_count", col("follower\_count").cast(IntegerType()))

df = df.withColumn("following\_count", col("following\_count").cast(IntegerType()))

df = df.withColumn("quote\_count", col("quote\_count").cast(IntegerType()))

df = df.withColumn("reply\_count", col("reply\_count").cast(IntegerType()))

df = df.withColumn("like\_count", col("like\_count").cast(IntegerType()))

df = df.withColumn("retweet\_count", col("retweet\_count").cast(IntegerType()))

df = df.withColumn("is\_retweet", col("is\_retweet").cast("boolean"))

print('Data frame type: ' + str(type(df)))

total\_twitter\_count = df.count()

print('Rows count:', total\_twitter\_count)

print('Data overview')

df.printSchema()

print('Columns overview')

pd.DataFrame(df.dtypes, columns = ['Column Name','Data type'])

def calculate\_statistics(df, columns):

statistics = {'Statistic': ['Mean', 'Median', 'Mode', 'Standard Deviation', 'Min', 'Max', 'Range', '25th Quantile', '75th Quantile', '95th Quartiles']}

for column in columns:

mode\_column = df.groupBy(column).count().sort(F.col("count").desc()).select(column).limit(1).collect()[0][0]

column\_values = [

df.agg({column: "avg"}).collect()[0][0], # среднее значение

df.approxQuantile(column, [0.5], 0.001)[0], # медиана

mode\_column, # мода

df.agg({column: "stddev"}).collect()[0][0], # стандартное отклонение

df.agg({column: 'min'}).collect()[0][0], # минимум

df.agg({column: 'max'}).collect()[0][0], # максимум

df.agg({column: 'max'}).collect()[0][0] - df.agg({column: 'min'}).collect()[0][0], # диапазон

df.approxQuantile(column, [0.25], 0.001)[0], # 25-й квантиль

df.approxQuantile(column, [0.75], 0.001)[0], # 50-й квантиль

df.approxQuantile(column, [0.95], 0.001)[0] # 75-й квантиль

]

statistics[f'{column}'] = column\_values

# Создаем DataFrame из словаря

statistics\_df = pd.DataFrame(statistics)

return statistics\_df

# Список столбцов для подсчета статистики

columns = ['follower\_count', 'following\_count', 'quote\_count', 'reply\_count', 'like\_count', 'retweet\_count']

# Получаем DataFrame со статистикой

result\_df = calculate\_statistics(df, columns)

# Выводим DataFrame

print(result\_df)

all\_tweet\_languages = df.groupBy("tweet\_language").count().limit(10).collect()

language\_followers\_statistic\_df = []

for row in all\_tweet\_languages:

tweet\_language = row["tweet\_language"]

count = row["count"]

stat\_dict = {'tweet\_language': tweet\_language, 'count': count}

language\_followers\_df = df.select(df.follower\_count).filter(df.tweet\_language == tweet\_language)

stats = language\_followers\_df.select(

F.percentile\_approx(df.follower\_count, [0.25, 0.5, 0.75, 0.95], 1000000).alias("percentiles"),

F.min(df.follower\_count).alias('min'),

F.max(df.follower\_count).alias('max'),

F.mean(df.follower\_count).alias('mean'),

F.stddev(df.follower\_count).alias('stddev')

).collect()[0]

stat\_dict['25th percentile'] = stats['percentiles'][0]

stat\_dict['50th percentile'] = stats['percentiles'][1]

stat\_dict['75th percentile'] = stats['percentiles'][2]

stat\_dict['95th percentile'] = stats['percentiles'][3]

stat\_dict['min'] = stats['min']

stat\_dict['max'] = stats['max']

stat\_dict['mean'] = stats['mean']

stat\_dict['stddev'] = stats['stddev']

language\_followers\_statistic\_df.append(stat\_dict)

language\_followers\_statistic\_df = pd.DataFrame(language\_followers\_statistic\_df)

language\_followers\_statistic\_df

follower\_counts = df.select('follower\_count').collect()

following\_counts = df.select('following\_count').collect()

reply\_counts = df.select('reply\_count').collect()

quote\_counts = df.select('quote\_count').collect()

like\_counts = df.select('like\_count').collect()

retweet\_counts = df.select('retweet\_count').collect()

# Построение диаграммы рассеяния

plt.figure(figsize=(11, 7))

plt.scatter([x['like\_count'] for x in like\_counts], [x['retweet\_count'] for x in retweet\_counts], alpha=0.3, color='skyblue')

plt.title('Диаграмма рассеивания количества лайков и количества ретвитов')

plt.xlabel('Количество лайков')

plt.ylabel('Количество ретвитов')

plt.show()

# Построение диаграммы рассеяния

plt.figure(figsize=(11, 7))

plt.scatter([x['reply\_count'] for x in reply\_counts], [x['quote\_count'] for x in quote\_counts], alpha=0.3, color='lightgreen')

plt.title('Диаграмма рассеивания количества ответов и количества цитирований')

plt.xlabel('Количество ответов')

plt.ylabel('Количество цитирований')

plt.show()

# Построение диаграммы рассеяния

plt.figure(figsize=(11, 7))

plt.scatter([x['follower\_count'] for x in follower\_counts], [x['following\_count'] for x in following\_counts], alpha=0.3, color='lightcoral')

plt.title('Диаграмма рассеивания подписок лайков и количества подписчиков')

plt.xlabel('Количество подписок')

plt.ylabel('Количество подписчиков')

plt.show()

# Построение диаграммы рассеяния

plt.figure(figsize=(11, 7))

plt.scatter([x['follower\_count'] for x in follower\_counts], [x['like\_count'] for x in like\_counts], alpha=0.3, color='orange')

plt.title('Диаграмма рассеивания количества подписчиков и количества лайков')

plt.xlabel('Количество подписчиков')

plt.ylabel('Количество лайков')

plt.show()

# Построение диаграммы рассеяния

plt.figure(figsize=(11, 7))

plt.scatter([x['like\_count'] for x in like\_counts], [x['reply\_count'] for x in reply\_counts], alpha=0.3, color='pink')

plt.title('Диаграмма рассеивания количества лайков и количества ответов')

plt.xlabel('Количество лайков')

plt.ylabel('Количество ответов')

plt.show()

**Приложение Б**

tp2 = float(newPrediction.filter("prediction == 1.0 AND label == 1").count())

fp2 = float(newPrediction.filter("prediction == 1.0 AND label == 0").count())

tn2 = float(newPrediction.filter("prediction == 0.0 AND label == 0").count())

fn2 = float(newPrediction.filter("prediction == 0.0 AND label == 1").count())

pr2 = tp2 / (tp2 + fp2)

re2 = tp2 / (tp2 + fn2)

metrics2 = spark.createDataFrame([

("TP", tp2),

("FP", fp2),

("TN", tn2),

("FN", fn2),

("Precision", pr2),

("Recall", re2),

("F1", 2\*pr2\*re2/(re2+pr2))],["metric", "value"])

metrics2.show()

evaluator2 = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction", metricName="areaUnderROC")

aur2 = evaluator2.evaluate(newPrediction)

print( "AUR2 = ", aur2)