

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Инженерная школа информационных технологий и робототехники
Направление подготовки 09.04.01 Информатика и вычислительная техника
Отделение Информационных технологий

Отчет по Практической работе №4
по дисциплине «Параллельные и высокопроизводительные вычисления»

Тема работы
Работа с фреймворком распределенных вычислений Apache Spark

Вариант 5

Студент

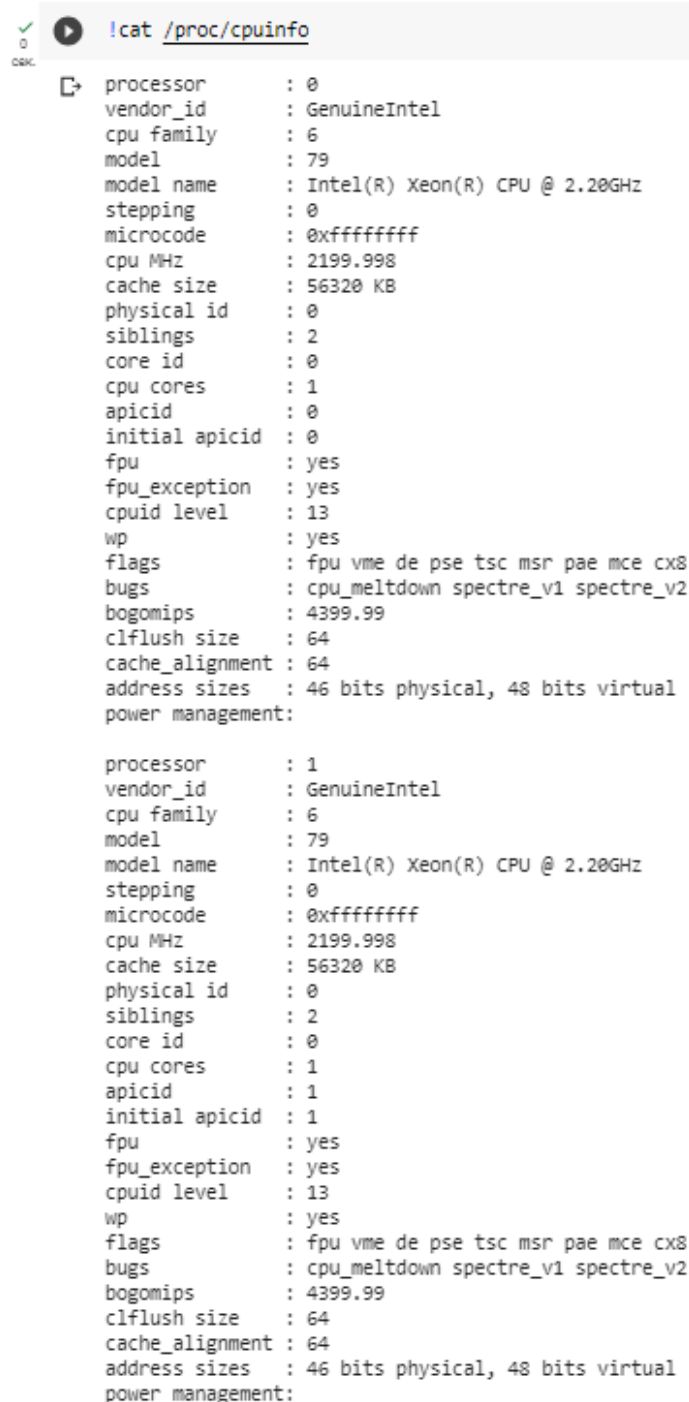
Группа	ФИО	Подпись	Дата
8BM22	Ямкин Н.Н.		

Руководитель

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ	Аксёнов С.В.	к.т.н., доцент		

Ход работы

Данная работа выполнялась на облачной платформе Google Colaboratory. Ниже приведены характеристики выделенных вычислительных ресурсов.



```
!cat /proc/cpuinfo

processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 79
model name     : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping       : 0
microcode      : 0xffffffff
cpu MHz        : 2199.998
cache size     : 56320 KB
physical id    : 0
siblings       : 2
core id        : 0
cpu cores      : 1
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8
bugs           : cpu_meltdown spectre_v1 spectre_v2
bogomips       : 4399.99
clflush size   : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id      : GenuineIntel
cpu family     : 6
model          : 79
model name     : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping       : 0
microcode      : 0xffffffff
cpu MHz        : 2199.998
cache size     : 56320 KB
physical id    : 0
siblings       : 2
core id        : 0
cpu cores      : 1
apicid         : 1
initial apicid : 1
fpu            : yes
fpu_exception  : yes
cpuid level    : 13
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8
bugs           : cpu_meltdown spectre_v1 spectre_v2
bogomips       : 4399.99
clflush size   : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

Рисунок 1 – Характеристики виртуальной машины

Apache Spark можно использовать на Google Colab без настроенного кластера, но в ограниченном режиме для одного рабочего узла.

По умолчанию Spark будет использовать только один исполнитель и один поток на машине. Это ограничение связано с ограниченными вычислительными ресурсами в Colab, где у вас есть только одна виртуальная машина доступная для выполнения кода.

Использование Spark в Colab всё равно предоставляет возможность применять функциональность Spark, такую как чтение и обработка данных, выполнение анализа данных, машинного обучения и многое другое. Однако для обработки больших объемов данных или сложных вычислений это может быть недостаточно.

Таким образом, Spark на Google Colab будет работать в локальном режиме, чтобы обеспечить небольшую функциональность Spark, которая может быть полезной для демонстрации, прототипирования или решения небольших задач.

Задание

Написать 3 программы на одном из языков программирования (Python, Java, Scala) с использованием инструментов фреймворка Apache Spark.

Программа А (работа с компонентом Spark SQL) представляет собой набор запросов к базе данных 'brooklyn_sales_map.csv'.

Программа В (работа с компонентом Spark MLlib) осуществляет построение трёх моделей машинного обучения: логистическая регрессия, дерево решений и случайный лес для набора данных, указанного в варианте. Для каждого алгоритма необходимо получить лучшую модель, путем поиска наилучшего набора её параметров. Для алгоритма логистической регрессии это параметры `maxIter=10...10000`, `regParam>0` (0.1, 0.5, 1, ...), `elasticNetParam=0...1`. Для дерева решений `maxDepth > 0` (3, 5, 9, 12, ...). Для случайного леса `maxDepth > 0` (3, 5, 9, 12, ...) и `numTree > 0` (5, 11, 25, ...). Получить значения матриц ошибок (Confusion Matrix), верности (Accuracy), полноты (Recall) и точности (Precision) для всех моделей.

Программа С (работа с компонентом Spark Core) осуществляет получение данных из набора данных FIFA World Cup 2018 Tweets, доступный

по адресу: <https://www.kaggle.com/datasets/rgupta09/world-cup-2018-tweets> и выполняет обработку данных набора с помощью устойчивых распределенных наборов данных RDD.

1 Программа А

1.1 Задание

- Найдите средний год постройки жилья (`year_built`) и выведите новую таблицу, содержащую год постройки жилья и отклонение года постройки от среднего значения.
- Подсчитайте, сколько различных домов приходится на каждую улицу, и выведите результат в отдельную таблицу.
- Отсортировать датасет по возрастанию цены продажи (`sale_price`) и убыванию индексов (`zip_code`) одновременно.
- Выведите таблицу с наибольшими ценами продажи (`sale_price`) и количеством зданий по каждому сочетанию соседства (`neighborhood`) и категории класса здания (`building_class_category`).

1.2 Листинг программы А

```
%%bash
apt-get install openjdk-17-jdk-headless -qq > /dev/null
%%bash
apt-get install openjdk-17-jdk-headless -qq > /dev/null
import os
os.environ['JAVA_HOME']='/usr/lib/jvm/java-1.17.0-openjdk-amd64'
os.environ['SPARK_HOME']='/content/spark-3.3.0-bin-hadoop3'

%%bash
pip install findspark

import findspark
findspark.init()

from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, RandomForestClassifier

from google.colab import drive
drive.mount('/content/gdrive')

#### Программа А

spark = SparkSession.builder\
    .master('local')\
    .appName("My Spark Session").getOrCreate()
df = spark.read.csv(r"/content/gdrive/MyDrive/brooklyn_sales_map.csv", header=True)
df.show()
```

Задание 1

```
year_built_mean = df.select('year_built').groupBy().agg(F.mean('year_built').alias('mean'))
year_built_mean_val = year_built_mean.collect()[0]['mean']
print("Средний год постройки жилья:", year_built_mean_val)

df_year_built_and_deviation = df.select('year_built')
df_year_built_and_deviation = df_year_built_and_deviation.withColumn('deviation, %',
(df_year_built_and_deviation.year_built - year_built_mean_val) / df_year_built_and_deviation.year_built * 100)
df_year_built_and_deviation.show()
```

Задание 2

```
df_street_building = df.groupBy('address9').agg(F.countDistinct('building_class_category').alias("Number of unique houses on the street"))
df_street_building.show()
```

Задание 3

```
# Сортировка датасета по возрастанию цены продажи и убыванию индексов
sorted_df = df.orderBy('sale_price', df['zip_code'].desc())
sorted_df.select('sale_price', 'zip_code').show()
```

Задание 4

```
# Группируем данные по соседству и категории класса здания и считаем максимальную цену продажи и количество зданий
sorted_df = df.groupBy('neighborhood', 'building_class_category')\
    .agg(F.max('sale_price').alias('max_sale_price'), F.count('*').alias('building_count'))\
    .orderBy('neighborhood', 'building_class_category')

# Закрытие сессии
spark.stop()
```

1.3 Результаты

year_built	deviation, %
2002	15.001682977138547
0	null
1924	11.555805259995516
1970	13.620999654939784
1927	11.693497312003826
0	null
1928	11.739299439954031
2012	15.424139821188554
0	null
1912	11.000716171669128
0	null
0	null
2009	15.297844360493466
1967	13.489257407336744
1920	11.37154652095384
1992	14.574984598509724
1920	11.37154652095384
2014	15.508127765755397
0	null
1962	13.268791702462474

only showing top 20 rows

Рисунок 2 – Таблица с годом постройки жилья и процентом его отклонения от среднего значения года постройки

address9	Number of unique houses on the street
119 NORTH 11TH ST...	0
6609 FORT HAMILTO...	1
784 4 AVENUE	1
8704-08 18 AVENUE	3
228 QUINCY STREET	1
285 PROSPECT PLAC...	1
1626-1628 UTICA A...	1
2700 VOORHIES AVENUE	1
999 EAST 108TH ST...	1
60 PINEAPPLE STRE...	1
547 49TH STREET	1
6614 14TH AVENUE	1
151 11TH STREET	1
400 EAST 17TH STR...	1
1577 EAST 17TH ST...	1
1081 FULTON STREET	1
66 EAST 52ND STREET	1
38 EAST 16TH STREET	1
938 PRESIDENT STR...	1
2041 WEST 7 STREET	1

only showing top 20 rows

Рисунок 3 – Таблица с количеством уникальных домов на каждой улице

sale_price	zip_code
0	33803
0	33803
0	11416
0	11416
0	11416
0	11416
0	11416
0	11416
0	11416
0	11416
0	11416
0	11416
0	11416
0	11249
0	11249
0	11249
0	11249
0	11249
0	11249
0	11249
0	11249

only showing top 20 rows

Рисунок 4 – Таблица с отсортированным по возрастанию цены продажи (sale_price) и убыванию индексов (zip_code) датасетом

neighborhood	building_count	max_sale_price
3004	2	346788
3019	1	0
BATH BEACH	6	0
BATH BEACH	509	9e+05
BATH BEACH	1808	9e+05
BATH BEACH	756	9e+05
BATH BEACH	450	753505
BATH BEACH	20	7e+05
BATH BEACH	28	960000
BATH BEACH	3	0
BATH BEACH	207	999988
BATH BEACH	11	850000
BATH BEACH	19	855421
BATH BEACH	406	94000
BATH BEACH	24	435000
BATH BEACH	102	98000
BATH BEACH	4	399000
BATH BEACH	69	550000
BATH BEACH	6	780000
BATH BEACH	71	840056

only showing top 20 rows

Рисунок 5 – Таблица с наибольшими ценами продажи (sale_price) и количеством зданий по каждому сочетанию соседства (neighborhood) и категории класса здания (building_class_category)

2 Программа В

2.1 Задание

Набор данных: Оценка вероятности пожара по детекторам физических характеристик среды:
<https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset> Метка класса: Fire_Alarm. Класс отрицательный – 0 (нет огня), класс положительный – 1 (пожар).

2.2 Листинг программы В

```
#### Программа В
spark = SparkSession.builder\
    .master('local')\
    .appName("My Spark Session").getOrCreate()
df = spark.read.csv(r"/content/gdrive/MyDrive/smoke_detection_iot.csv",header=True)
df.show()

new_column_names = ['_c0','UTC','Temperature[C]','Humidity[%]','TVOC[ppb]','eCO2[ppm]','Raw H2','Raw
Ethanol','Pressure[hPa]','PM10','PM25','NC05','NC10','NC25','CNT','Fire Alarm']
df = df.toDF(*new_column_names)
df.columns

feature_columns = ['_c0','UTC','Temperature[C]','Humidity[%]','TVOC[ppb]','eCO2[ppm]','Raw H2','Raw
Ethanol','Pressure[hPa]','PM10','PM25','NC05','NC10','NC25','CNT']
target_column = ['Fire Alarm']

all_columns = feature_columns + target_column

string_col_2_float = ['PM10', 'NC10', 'PM25', 'NC05', 'NC25', 'Humidity[%]', 'Pressure[hPa]', 'Temperature[C]']

# Из string в numeric
for column in all_columns:
    if column in string_col_2_float:
        df = df.withColumn(column, F.col(column).cast('Double'))
    else:
        df = df.withColumn(column, F.col(column).cast('Integer'))
```

```

def df_2_vector(dataframe, inputCols, outputCol):
    assembler = VectorAssembler(
        inputCols=inputCols, outputCol=outputCol)
    assembled_df = assembler.transform(dataframe)
    return assembled_df.randomSplit([0.8, 0.2])

cols = feature_columns
train_df, test_df = df_2_vector(df, cols, 'vectorized_data')

#### Логистическая регрессия

logistic_regression_model = LogisticRegression(labelCol= 'Fire Alarm', featuresCol = 'vectorized_data',
maxIter=10000, regParam=0.1, elasticNetParam=0.8)
model_lr = logistic_regression_model.fit(train_df)

predictions_lr = model_lr.transform(test_df)

predictions_lr.select('Fire Alarm','probability','prediction').show(truncate=False)

print('Модель Логистической регрессии. Правильные предсказания')
TP = predictions_lr[(predictions_lr['Fire Alarm']==1)&(predictions_lr['prediction']==1)].count()
print('Количество верно идентифицированных угроз пожарной тревоги (правильное предсказание):',TP)
TN = predictions_lr[(predictions_lr['Fire Alarm']==0)&(predictions_lr['prediction']==0)].count()
print('Количество верно идентифицированных не угроз пожарной тревоги (правильное предсказание):',TN)

print('Модель Логистической регрессии. Ошибки')
FP = predictions_lr[(predictions_lr['Fire Alarm']==0)&(predictions_lr['prediction']==1)].count()
print('Количество ложно идентифицированных не угроз пожарной тревоги (ошибочное предсказание):',FP)
FN = predictions_lr[(predictions_lr['Fire Alarm']==1)&(predictions_lr['prediction']==0)].count()
print('Количество ложно идентифицированных угроз пожарной тревоги (ошибочное предсказание):',FN)

```

```

#Метрики качества моделей
#Accuracy - Верность Acc. = (TP+TN)/(TP+TN+FP+FN)
acc_lr = (TP+TN)/(TP+TN+FP+FN)
print('Верность модели (логистическая регрессия):', round(acc_lr,2))

#Precision - Точность Prec. = TP/(TP+FP)
precision_lr = TP/(TP+FP)
print('Точность модели (логистическая регрессия):', round(precision_lr,2))

#Recall - Полнота Recall = TP/(TP+FN)
recall_lr = TP/(TP+FN)
print('Полнота модели (логистическая регрессия):', round(recall_lr,2))

#### Дерево решений

#Decision Tree Model
tree_model = DecisionTreeClassifier(featuresCol='vectorized_data',
                                   labelCol='Fire Alarm',maxDepth=3).fit(train_df)

tree_predictions = tree_model.transform(test_df)

tree_predictions.select('Fire Alarm','probability','prediction').show(truncate=False)

print('Модель Деревя решений. Правильные предсказания')
TP = tree_predictions[(tree_predictions['Fire Alarm']==1)&(tree_predictions['prediction']==1)].count()
print('Количество верно идентифицированных угроз пожарной тревоги (правильное предсказание):',TP)
TN = tree_predictions[(tree_predictions['Fire Alarm']==0)&(tree_predictions['prediction']==0)].count()
print('Количество верно идентифицированных не угроз пожарной тревоги (правильное предсказание):',TN)

print('Модель Деревя решений. Ошибки')
FP = tree_predictions[(tree_predictions['Fire Alarm']==0)&(tree_predictions['prediction']==1)].count()

```

```

print('Количество ложно идентифицированных не угроз пожарной тревоги (ошибочное предсказание):', FP)
FN = tree_predictions[(tree_predictions['Fire Alarm']==1)&(tree_predictions['prediction']==0)].count()
print('Количество ложно идентифицированных угроз пожарной тревоги (ошибочное предсказание):', FN)

#Метрики качества моделей
#Accuracy - Верность Acc. = (TP+TN)/(TP+TN+FP+FN)
acc_tree = (TP+TN)/(TP+TN+FP+FN)
print('Верность модели (дерево решений):', round(acc_tree,2))

#Precision - Точность Prec. = TP/(TP+FP)
precision_tree = TP/(TP+FP)
print('Точность модели (дерево решений):', round(precision_tree,2))

#Recall - Полнота Recall = TP/(TP+FN)
recall_tree = TP/(TP+FN)
print('Полнота модели (дерево решений):', round(recall_tree,2))

#### Случайный лес

rf_model = RandomForestClassifier(featuresCol='vectorized_data',
                                labelCol='Fire Alarm',maxDepth=5, numTrees=5).fit(train_df)

rf_predictions = rf_model.transform(test_df)
rf_predictions.select('Fire Alarm','probability','prediction').show(truncate=False)

print('Модель Случайного леса. Правильные предсказания')
TP = rf_predictions[(rf_predictions['Fire Alarm']==1)&(rf_predictions['prediction']==1)].count()
print('Количество верно идентифицированных угроз пожарной тревоги (правильное предсказание):', TP)
TN = rf_predictions[(rf_predictions['Fire Alarm']==0)&(rf_predictions['prediction']==0)].count()
print('Количество верно идентифицированных не угроз пожарной тревоги (правильное предсказание):', TN)

```

```

print('Модель Случайного леса. Ошибки')
FP = rf_predictions[(rf_predictions['Fire Alarm']==0)&(rf_predictions['prediction']==1)].count()
print('Количество ложно идентифицированных не угроз пожарной тревоги (ошибочное предсказание):',FP)
FN = rf_predictions[(rf_predictions['Fire Alarm']==1)&(rf_predictions['prediction']==0)].count()
print('Количество ложно идентифицированных угроз пожарной тревоги (ошибочное предсказание):',FN)

#Метрики качества моделей
#Accuracy - Верность Acc. = (TP+TN)/(TP+TN+FP+FN)
acc_rf = (TP+TN)/(TP+TN+FP+FN)
print('Верность модели (случайный лес):', round(acc_rf,3))

#Precision - Точность Prec. = TP/(TP+FP)
precision_rf = TP/(TP+FP)
print('Точность модели (случайный лес):', round(precision_rf,3))

#Recall - Полнота Recall = TP/(TP+FN)
recall_rf = TP/(TP+FN)
print('Полнота модели (случайный лес):', round(recall_rf,3))

# Заккрытие сессии
spark.stop()

```

2.3 Результаты

Модель Логистической регрессии. Правильные предсказания
Количество верно идентифицированных угроз пожарной тревоги (правильное предсказание): 8770
Количество верно идентифицированных не угроз пожарной тревоги (правильное предсказание): 2794

Модель Логистической регрессии. Ошибки
Количество ложно идентифицированных не угроз пожарной тревоги (ошибочное предсказание): 737
Количество ложно идентифицированных угроз пожарной тревоги (ошибочное предсказание): 236

Верность модели (логистическая регрессия): 0.92
Точность модели (логистическая регрессия): 0.92
Полнота модели (логистическая регрессия): 0.97

Рисунок 6 – Метрики качества модели логистической регрессии при следующих параметрах maxIter=10000, regParam=0.1, elasticNetParam=0.8

Модель Древа решений. Правильные предсказания
Количество верно идентифицированных угроз пожарной тревоги (правильное предсказание): 8716
Количество верно идентифицированных не угроз пожарной тревоги (правильное предсказание): 3486

Модель Древа решений. Ошибки
Количество ложно идентифицированных не угроз пожарной тревоги (ошибочное предсказание): 85
Количество ложно идентифицированных угроз пожарной тревоги (ошибочное предсказание): 238

Верность модели (дерево решений): 0.97
Точность модели (дерево решений): 0.99
Полнота модели (дерево решений): 0.97

Рисунок 7 – Метрики качества модели дерева решений при параметре maxDepth = 3

Модель Случайного леса. Правильные предсказания
Количество верно идентифицированных угроз пожарной тревоги (правильное предсказание): 8954
Количество верно идентифицированных не угроз пожарной тревоги (правильное предсказание): 3566

Модель Случайного леса. Ошибки
Количество ложно идентифицированных не угроз пожарной тревоги (ошибочное предсказание): 5
Количество ложно идентифицированных угроз пожарной тревоги (ошибочное предсказание): 0

Верность модели (случайный лес): 0.9996
Точность модели (случайный лес): 0.9994
Полнота модели (случайный лес): 1.0

Рисунок 8 – Метрики качества модели случайного леса при параметре maxDepth = 5, numTrees = 5

3 Программа С

3.1 Задание

Получить десять наиболее упоминаемых хештегов (датафрейм **А**). Создать отдельный файл со списком столиц всех государств. Из этого списка выделить датафрейм **Б**, содержащий десять столиц, из которых твиты отправлялись наиболее часто. Выделить, в каких столицах из датафрейма **Б** самый часто используемый хештег принадлежит датафрейму **А**.

3.2 Листинг программы С

```
from itertools import islice
from pyspark.sql.functions import col
from pyspark import SparkContext

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

# Создание SparkContext
sc = spark.sparkContext

def take(n, iterable):
    return list(islice(iterable, n))

def sort_dict_by_value(dict, reverse=True):
    return {key: value for key, value in sorted(dict.items(), key=lambda x: x[1], reverse=reverse)}

main_df = spark.read.csv('/content/gdrive/MyDrive/FIFA.csv', header=True, multiLine=True, escape="\\"")

#### 10 самых популярных хештегов

groupedHashTagsByCount = main_df.rdd \
    .filter(lambda x: x['Hashtags'] != None) \
    .flatMap(lambda x: x['Hashtags'].split(',')) \
    .countByValue()

groupedHashTagsByCountSorted = sort_dict_by_value(groupedHashTagsByCount)
top10Hashtags = take(10, groupedHashTagsByCountSorted.items())
top10HashtagsDf = spark.createDataFrame(data=top10Hashtags, schema = ["hashtag", "count"])
print('Результат')
top10HashtagsDf.show()
```



```

##### 10 самых популярных столиц, из которых отправляли твиты

capitals_df = spark.read.csv('country-list.csv', header=True, multiLine=True, escape="\")
capitals_df = capitals_df.drop('country', 'type')

new_df = main_df.join(capitals_df, capitals_df.capital == main_df.Place, "leftouter")
groupedCapitalsByCount = new_df.rdd\
    .filter(lambda x: x['capital'] != None)\
    .map(lambda x: x['Place'])\
    .countByValue()\

groupedCapitalsByCountSorted = sort_dict_by_value(groupedCapitalsByCount)
top10Capitals = take(10, groupedCapitalsByCountSorted.items())
top10CapitalsDf = spark\
    .createDataFrame(data=top10Capitals, schema = ["capital", "count"])
print('Результат')
top10CapitalsDf.show()

##### Столицы, с самыми популярными хештегами в топ 10 хештегов

new_df = new_df\
    .filter(col('capital').isNotNull())\
    .drop(col('capital'))\
    .join(top10CapitalsDf, top10CapitalsDf.capital == new_df.Place, "leftouter")

capitalhashTagDf = new_df.rdd\
    .filter(lambda x: x['capital'] != None) \
    .filter(lambda x: x['Hashtags'] != None) \
    .flatMap(lambda x: [(x['capital'], hashtag) for hashtag in x['Hashtags'].split(',')])\
    .toDF()\
    .filter(col('_2') != 'WorldCup')

```

```

countedCapitalHashTag = capitalhashTagDf.groupBy('_1', '_2').count()

counted = countedCapitalHashTag\
.groupBy('_1')\
.agg(F.max('count').alias('num_of_hashtags'))\
.withColumnRenamed("_1", "capital")

join_conditions = [countedCapitalHashTag['_1'] == counted['capital'], countedCapitalHashTag['count'] ==
counted['num_of_hashtags']]
topCapitalsHashTag = counted \
.join(countedCapitalHashTag, join_conditions) \
.filter(col('num_of_hashtags').isNotNull()) \
.drop(col('_1')) \
.drop(col('count'))

topCapitalsHashtagsWhichPopular = topCapitalsHashTag \
.join(top10HashtagsDf, \
      top10HashtagsDf['hashTag'] == topCapitalsHashTag['_2']) \
.filter(col('hashTag').isNotNull()) \
.drop(col('count')) \
.drop(col('_2'))

print('Результат')
topCapitalsHashtagsWhichPopular.show()

```

3.3 Результаты

hashtag	count
WorldCup	239007
FRA	31717
FRAARG	21408
ENG	19459
worldcup	18897
FRABEL	18810
CRO	15819
ARG	15172
FIFASTadiumDJ	14198
CRODEN	12478

Рисунок 9 – 10 наиболее упоминаемых хэштегов

capital	count
London	2212
Singapore	514
Nairobi	390
Paris	233
New Delhi	196
Abuja	188
Kuala Lumpur	162
Dublin	159
Jakarta	136
Accra	111

Рисунок 10 – 10 столиц, из которых твиты отправлялись наиболее часто

capital	num_of_hashtags	hashtag
New Delhi	27	FRA
Nairobi	55	FRA
Accra	13	FRA
Kuala Lumpur	31	FRA
Jakarta	29	FRA
Singapore	87	FRA
London	308	ENG
Dublin	21	worldcup
Paris	55	FRAARG
Abuja	45	FRAARG

Рисунок 11 – Столицы, у которых самый популярный хэштег относится к 10 самым популярным хэштегам в целом

4 Вывод

В ходе выполнения лабораторной работы были реализованы программы с использованием фреймворка Apache Spark на языке Python. Были изучены и применены на практике следующие компоненты фреймворка: Spark SQL, Spark Mllib и Spark Core.