

Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики управления и технологий

Кузьмина Дарья Юрьевна БД-241м

Инструменты хранения и анализа больших данных

Лабораторная работа 1.2
Обработка данных с использованием Apache Spark и Python (PySpark)
Вариант 11

Направление подготовки/специальность
38.04.05 - Бизнес-информатика
Бизнес-аналитика и большие данные
(очная форма обучения)

Руководитель дисциплины:
Босенко Т.М., доцент департамента
информатики, управления и технологий,
доктор экономических наук

Москва
2025

Содержание

Введение	2
Основная часть	2
Заключение	21

Введение

Цель

освоение основ работы с Apache Spark и его интеграцией с Python через библиотеку PySpark. Студенты научатся обрабатывать большие объемы данных, используя распределенные вычисления, а также научатся применять базовые операции с RDD (Resilient Distributed Datasets) и DataFrame, работать с SQL-запросами в Spark SQL, а также визуализировать результаты обработки данных.

Задачи:

1. Установить Apache Spark и PySpark.
Настроить рабочую среду для использования PySpark в Python, установить необходимые зависимости и настроить Spark на локальном компьютере или через облачную платформу.
2. Загрузка данных и их предварительная обработка.
Скачать или подготовить исходные данные для анализа (например, текстовые файлы или CSV). Загружать данные в Spark через RDD или DataFrame, выполнить предварительную обработку: очистка данных, фильтрация, преобразования.
3. Применение операций с RDD и DataFrame.
Научиться работать с RDD и DataFrame, выполнять такие операции как map, filter, reduce, groupBy, join и другие стандартные операции для обработки данных в распределенной среде.
4. Применение SQL-запросов через Spark SQL.
Использование SQL-запросов в Spark для извлечения и агрегации данных, создание временных таблиц и выполнение сложных запросов для анализа данных.
5. Визуализация результатов анализа данных.
Визуализировать полученные результаты с помощью библиотеки Python для визуализации данных (например, matplotlib или seaborn). Построить графики для лучшего представления результатов.
6. Подготовка отчета.
Оформить отчет по выполненной практике, в котором будет описан процесс выполнения работы, анализ полученных результатов и выводы. Включить ссылки на репозиторий и прикрепить сам отчет в формате PDF или Markdown.

Основная часть

● **hadoop@devopsvm**:~/Downloads/lab1.2\$ start-dfs.sh

Starting namenodes on [localhost]

Starting datanodes

Starting secondary namenodes [devopsvm]

2025-04-22 00:44:23,975 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

○ **hadoop@devopsvm**:~/Downloads/lab1.2\$

● **hadoop@devopsvm**:~/Downloads/lab1.2\$ start-yarn.sh

Starting resourcemanager

Starting nodemanagers

○ **hadoop@devopsvm**:~/Downloads/lab1.2\$

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities ▾

Browse Directory

/

Go!

Show

25 ▾

entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Aug 26 2024	0	0 B	user
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Oct 17 2024	0	0 B	user2

Showing 1 to 2 of 2 entries

Previous

1

Next



Cluster

[About](#)
[Nodes](#)
[Node Labels](#)
[Applications](#)
NEW
NEW SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
[Scheduler](#)

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed
0	0	0	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes
1	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type
Capacity Scheduler	[memory-mb (unit=Mi), vcores]

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime
Showing 0 to 0 of 0 entries							

```
hadoop@devopsvm:~/Downloads/lab1.2$ hdfs dfs -mkdir -p /userkuz/sparkdir
2025-04-22 00:55:01,207 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

Show 25 entries

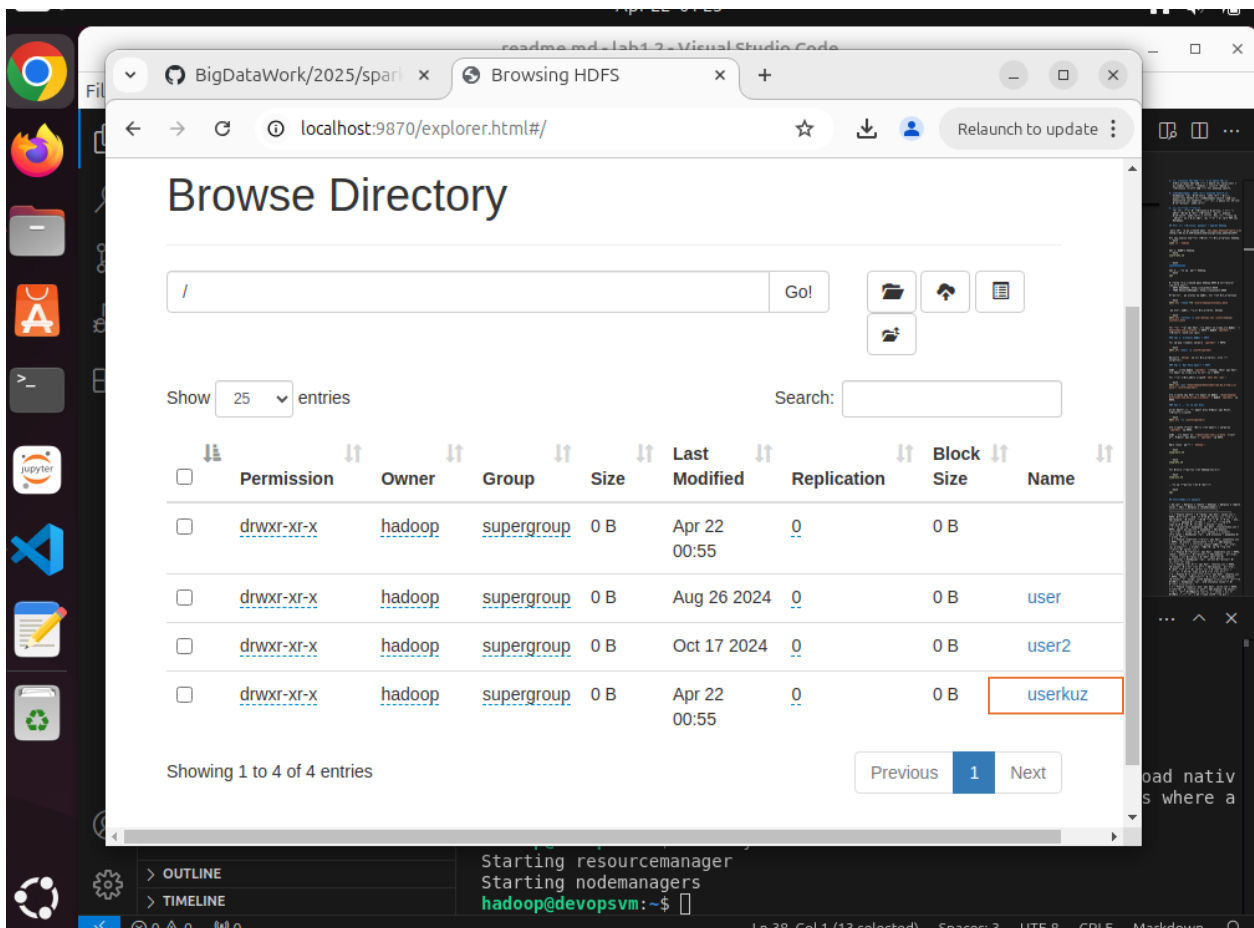
Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Apr 22 00:55	0	0 B	
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Aug 26 2024	0	0 B	user
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Oct 17 2024	0	0 B	user2
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Apr 22 00:55	0	0 B	userkuz

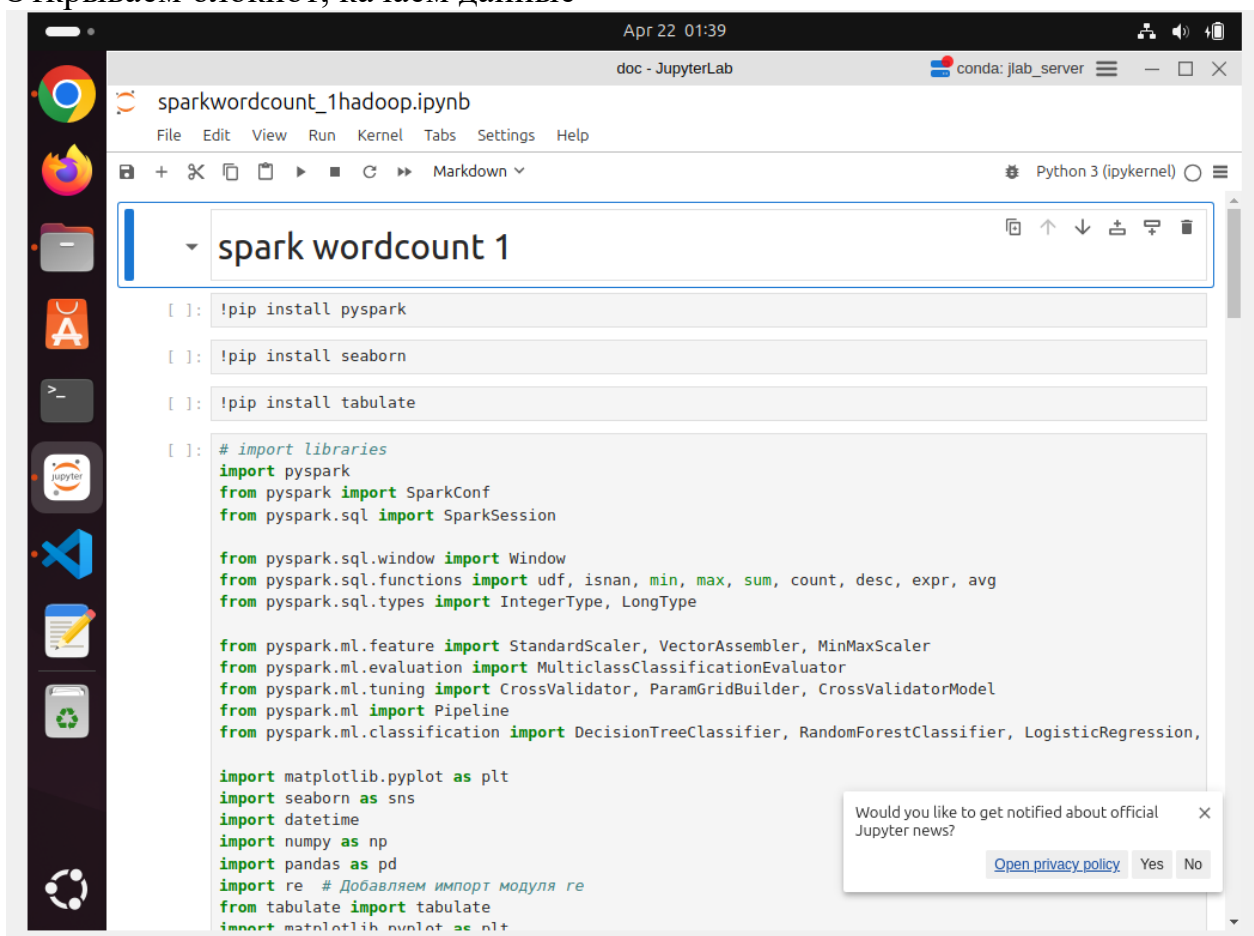
Showing 1 to 4 of 4 entries

Previous1Next

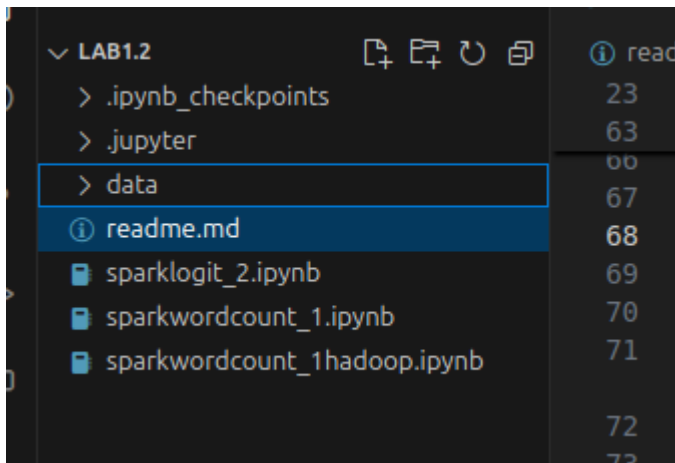
Понимаем, что на hadoop не установлены среда разработки для питона и идем с лицом лягухи в devops, чтобы оттуда уже выйти в hadoop, повторяя все вышеперечисленные команды.



Открываем блокнот, качаем данные



Скачиваем data



Создаем источник данных и загружаем туда данные

```
hadoop@devopsvm:~$ cd Downloads
hadoop@devopsvm:~/Downloads$ ls
2025
hadoop@devopsvm:~/Downloads$ cd ..
hadoop@devopsvm:~$ hdfs dfs -put /home/hadoop/Downloads/2025/data/* /userkuz/sparkdir/
2025-04-22 03:04:53,923 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
```

Browse Directory

Show

25

 entries

Search:

<input type="checkbox"/>	<div>↓↑</div> Permission	<div>↓↑</div> Owner	<div>↓↑</div> Group	<div>↓↑</div> Size	<div>↓↑</div> Last Modified	<div>↓↑</div> Replication	<div>↓↑</div> Block Size	<div>↓↑</div> Name
<input type="checkbox"/>	drwxr-xr-x	hadoop	supergroup	0 B	Apr 22 00:55	0	0 B	sparkdir

Showing 1 to 1 of 1 entries

Previous



1

Next

Browse Directory














/userkuz/sparkdir

Go!



Show 25 entries

Search:

<input type="checkbox"/>	Permissions	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	847.88 KB	Apr 22 03:05	1	128 MB	access.log	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	197.95 MB	Apr 22 03:05	1	128 MB	brooklyn_sales_map.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	483.13 KB	Apr 22 03:05	1	128 MB	digits.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	222.08 MB	Apr 22 03:05	1	128 MB	food-inspections.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	4.47 KB	Apr 22 03:05	1	128 MB	idiot.txt	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	6.44 KB	Apr 22 03:05	1	128 MB	ip.tsv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	2.85 KB	Apr 22 03:05	1	128 MB	kmeans.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	2.39 KB	Apr 22 03:05	1	128 MB	logit_test.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	612 B	Apr 22 03:05	1	128 MB	logit_train.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	2.33 MB	Apr 22 03:05	1	128 MB	ratings.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	3.45 MB	Apr 22 03:05	1	128 MB	reviews_sample.json	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	120.63 KB	Apr 22 03:05	1	128 MB	sales.csv	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	114 B	Apr 22 03:05	1	128 MB	sales_header.csv	

```
sparkwordcount_1hadoop.ipynb
File Edit View Run Kernel Tabs Settings Help

[5]: from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("WordCount App") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://localhost:9000") \
    .config("spark.ui.port", "4050") \
    .getOrCreate()

# Установка количества разделов для shuffle операций
spark.conf.set("spark.sql.shuffle.partitions", "50")

# Чтение данных из HDFS (текстовый файл)
file_path = "hdfs://localhost:9000/userkuz/sparkdir/idiot.txt"
df = spark.read.text(file_path) # Используем .text, так как это текстовый файл

# Печать первых нескольких строк
df.show()
```

25/04/22 03:14:44 WARN Utils: Your hostname, devopsvm resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)

25/04/22 03:14:45 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

25/04/22 03:14:50 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

```
+-----+
|               value|
+-----+
|Beauty will save ...|
|Don't let us forg...|
|It is better to b...|
|There is somethin...|
|Lack of originali...|
```

Would you like to get notified about official Jupyter news?

sparkwordcount_1hadoop.ipynb

File Edit View Run Kernel Tabs Settings Help

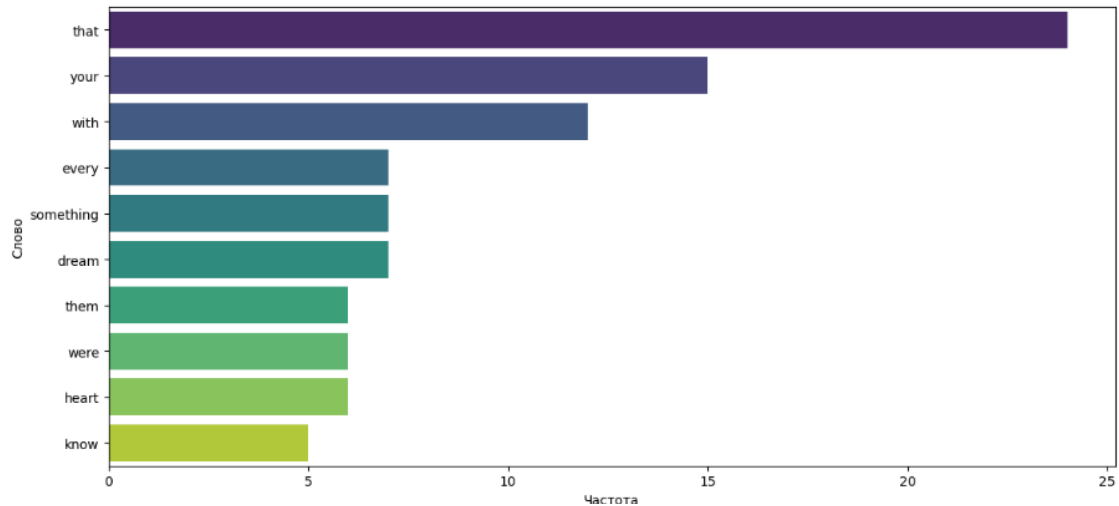
+ ✂ 📄 ▶ ■ ↺ ⏭ Code ▾

Python 3 (ipykernel) ○ ≡

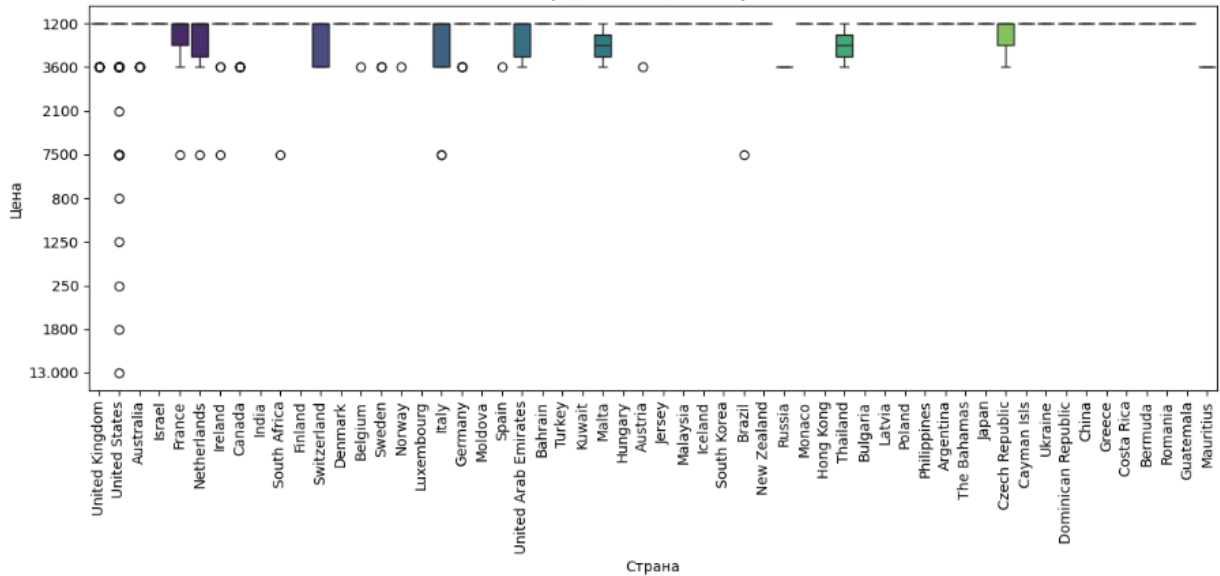
Топ-10 наиболее частых слов:

word	count
that	24
your	15
with	12
every	7
something	7
dream	7
them	6
were	6
heart	6
know	5

Топ-10 наиболее часто встречающихся слов



Распределение цен по странам




```
[11]: # TODO: Вывод дохода по стране и штату
revenue_by_country_state = spark.sql("""
    SELECT Country, State, SUM(Price) as Total_Revenue
    FROM sales
    GROUP BY Country, State
    ORDER BY Total_Revenue DESC
    """)

# Отображаем результаты
revenue_by_country_state.show()
```

```
+-----+-----+-----+
|      Country|      State|Total_Revenue|
+-----+-----+-----+
|United Kingdom|      England|    120000.0|
|United States|         CA|    113350.0|
|United States|         NY|     61200.0|
|United States|         TX|     55500.0|
|United States|         FL|     51600.0|
|      Canada|      Ontario|     46800.0|
|United States|         VA|     40400.0|
|      Canada|British Columbia|     28800.0|
|      Ireland|         Dublin|     28800.0|
|United States|         GA|     28200.0|
|      Canada|      Alberta|     26400.0|
|United States|         WA|     24000.0|
|United States|         IL|     24000.0|
|Netherlands|Zuid-Holland|     23100.0|
|United States|         NJ|     22800.0|
|United States|         MD|     22800.0|
|Australia|New South Wales|     20400.0|
|United States|         PA|     20400.0|
|United States|         TN|     19500.0|
|United States|         MN|     19200.0|
+-----+-----+-----+
```

Видим результат выполнения программы

Индивидуальное задание

11	Анализ доставки: загрузить delivery.csv в HDFS, оценить эффективность логистики	SQL-анализ: рассчитать стоимость доставки по регионам и типам доставки	Построить карту загруженности по регионам
----	---	--	---

Создадим новую директорию под задание

```
hadoop@devopsvm:~$ hdfs dfs -put /home/hadoop/Downloads/delivery.csv /userkd/var11/
2025-04-22 23:21:16,928 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$
```

Загрузим данные

Browse Directory

Show

25

 entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	7.05 KB	Apr 22 23:21	1	128 MB	delivery.csv	

Showing 1 to 1 of 1 entries

Previous

1

Next

Откроем ноутбук, проверим подключение.

sparksql

```
[7]: from pyspark.sql import SparkSession
```

```
# Создание SparkSession
spark = SparkSession.builder \
    .appName("SQL App") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://localhost:9000") \
    .config("spark.ui.port", "4050") \
    .getOrCreate()

# Установка количества разделов для shuffle операций
spark.conf.set("spark.sql.shuffle.partitions", "50")

# Чтение данных из HDFS (текстовый файл)
file_path = "hdfs://localhost:9000/userkuz/sparkdir/sales.csv"
header_path = "hdfs://localhost:9000/userkuz/sparkdir/sales_header.csv"
```

```
25/04/22 03:16:13 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
```

```
[8]: def parse_row(line):
      # Парсим строку (разделение по запятой)
      return line.split(',')

```

```
[6]: df.show()
```

	_c0	_c1	_c2	_c3	_c4	_c5	_c6
delivery_id	date	origin	destination	distance_km	delivery_time_min	status	
1	2025-04-13	Москва	Екатеринбург	1473	738	delayed	
2	2025-04-18	Нижний Новгород	Воронеж	724	415	delayed	
3	2025-04-01	Санкт-Петербург	Самара	980	647	delivered	
4	2025-04-08	Казань	Самара	231	122	delivered	
5	2025-04-21	Санкт-Петербург	Казань	883	570	delivered	
6	2025-04-10	Екатеринбург	Самара	115	47	delivered	
7	2025-04-05	Екатеринбург	Казань	1211	501	delivered	
8	2025-04-19	Самара	Саратов	1194	693	delivered	
9	2025-04-20	Воронеж	Санкт-Петербург	327	151	canceled	
10	2025-04-10	Санкт-Петербург	Воронеж	1381	825	delivered	
11	2025-04-18	Москва	Санкт-Петербург	1375	850	delivered	
12	2025-04-13	Самара	Санкт-Петербург	751	367	delivered	
13	2025-04-11	Нижний Новгород	Казань	206	104	delayed	
14	2025-04-03	Москва	Нижний Новгород	667	344	delivered	
15	2025-04-06	Екатеринбург	Санкт-Петербург	204	125	delivered	
16	2025-04-01	Екатеринбург	Санкт-Петербург	1314	733	delivered	
17	2025-04-21	Екатеринбург	Нижний Новгород	659	310	delivered	
18	2025-04-20	Нижний Новгород	Саратов	939	574	delivered	
19	2025-04-14	Самара	Нижний Новгород	699	318	delivered	

only showing top 20 rows

```
[9]: # Посмотрим схему
df.printSchema()
```

```
root
|-- _c0: string (nullable = true)
|-- _c1: string (nullable = true)
|-- _c2: string (nullable = true)
|-- _c3: string (nullable = true)
|-- _c4: string (nullable = true)
|-- _c5: string (nullable = true)
|-- _c6: string (nullable = true)
```

Посмотрим схему.

Результат выполнения задания.

```
:
from pyspark.sql.functions import col, avg

# Среднее время доставки (только для доставленных)
df.filter(col("_c6") == "delivered") \
  .agg(avg(col("_c5").cast("double")).alias("avg_delivery_time")) \
  .show()

[Stage 12:>                                     (0 + 1) / 1]
+-----+
|avg_delivery_time|
+-----+
|449.6865671641791|
+-----+

# Процент доставленных заказов
total = df.count()
delivered = df.filter(col("_c6") == "delivered").count()
print(f"Процент доставленных: {delivered / total * 100:.2f}%")

[Stage 18:===== (1 + 0) / 1]
Процент доставленных: 66.34%

# Эффективность: мин/км
df = df.withColumn("min_per_km",
                  col("_c5").cast("double") /
                  col("_c4").cast("double"))

df.filter(col("_c6") == "delivered") \
  .agg(avg("min_per_km").alias("avg_min_per_km")) \
  .show()

[Stage 21:>                                     (0 + 1) / 1]
+-----+
|   avg_min_per_km|
+-----+
|0.5145916386308724|
+-----+

11/
2025-04-22 23:21:16,928 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
hadoop@devopsvm:~$ stop-all.sh
WARNING: Stopping all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: Use CTRL-C to abort.
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [devopsvm]
2025-04-23 00:04:50,193 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Stopping nodemanagers
localhost: WARNING: nodemanager did not stop gracefully after 5 seconds: Trying t
o kill with kill -9
Stopping resourcemanager
hadoop@devopsvm:~$ jps
12248 Jps
hadoop@devopsvm:~$
```

Потушили hadoop.

colab.research.google.com

BigDataWork/2025/sparklogit_2.ipynb at main · Bos...

colab.google

sparklogit_2.ipynb

Файл Изменить Вид Вставка Среда выполнения Инструменты Справка

Команды + Код + Текст

Файлы

- sample_data
- delivery.csv

Как работает логистическая регрессия в Spark: особенности прогноза

Логистическая регрессия – это статистическая модель, которая используется в машинном обучении для прогнозирования вероятности возникновения некоторого события путем построения логистической функции и сравнения этого события с кривой этой функции. В результате формируется ответ в виде вероятности бинарного события: 0 и 1, где 0 – событие не произошло, 1 – событие произошло.

Работа с логистической регрессией в Spark

Для того, чтобы начать работу по прогнозу данных, необходимо настроить базовую конфигурацию, импортировав некоторые классы библиотек Spark MLlib и Spark SQL:

```
[ ] !pip install pyspark
[ ] Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.5.0)
[ ] Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)

[ ] !pip install findspark
[ ] Requirement already satisfied: findspark in /usr/local/lib/python3.10/dist-packages (2.0.1)

[ ] from pyspark.ml import Pipeline
[ ] from pyspark.ml.classification import LogisticRegression
[ ] from pyspark.ml.feature import HashingTF, Tokenizer
[ ] from pyspark.sql.functions import UserDefinedFunction
[ ] from pyspark.sql.types import *
```

ДАТАСЕТ С ДОМАМИ НА ПРОДАЖУ

В качестве примера мы будем использовать датасет Kaggle, который содержит данные о домах на продажу в Бруклине с 2003 по 2017 года и доступен для скачивания. Он содержит 111 атрибутов (столбцов) и 390883 записей (строк). В атрибуты включены: дата продажи, дата постройки, цена на дом, налоговый класс, соседние регионы, долгота, ширина и др.

```
[ ] from google.colab import drive
[ ] drive.mount('/content/drive')
```

Открыли файл, скачали данные

Как работает логистическая регрессия в Spark: особенности прогноза

Логистическая регрессия – это статистическая модель, которая используется в машинном обучении для прогнозирования вероятности возникновения некоторого события путем построения логистической функции и сравнения этого события с кривой этой функции. В результате формируется ответ в виде вероятности бинарного события: 0 и 1, где 0 – событие не произошло, 1 – событие произошло.

Работа с логистической регрессией в Spark

Для того, чтобы начать работу по прогнозу данных, необходимо настроить базовую конфигурацию, импортировав некоторые классы библиотек Spark MLlib и Spark SQL:

```
[1] !pip install pyspark
3
ЕК. Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
```

```
[2] !pip install findspark
7
ЕК. Collecting findspark
Downloading findspark-2.0.1-py2.py3-none-any.whl.metadata (352 bytes)
Downloading findspark-2.0.1-py2.py3-none-any.whl (4.4 kB)
Installing collected packages: findspark
Successfully installed findspark-2.0.1
```

```
[3] from pyspark.ml import Pipeline
2
from pyspark.ml.classification import LogisticRegression
ЕК. from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *
```

ДАТАСЕТ С ДОМАМИ НА ПРОДАЖУ

В качестве примера мы будем использовать датасет Kaggle, который содержит данные о домах на продажу в Бруклине с 2003 по 2017 года и доступен для скачивания. Он содержит 111 атрибутов (столбцов) и 390883 записей (строк). В атрибуты включены: дата продажи, дата постройки, цена на дом, налоговый класс, соседние регионы, долгота, ширина и др.

```
[ ] from google.colab import drive
7
drive.mount('/content/drive')
ЕК. Mounted at /content/drive
```

Подключились к диску

```
✓ [4] from google.colab import drive
17 drive.mount('/content/drive')
сек. ⤵ Mounted at /content/drive
```

```
✓ [6] import os
0
сек. ▶ os.chdir("/content/drive/My Drive/Colab Notebooks/Var11")
0 os.listdir()
сек. ⤵ ['delivery.csv']
```

В директории больше файлов нет

```
▶ from pyspark.sql import functions as F
# Бинаризация статуса
data = data.withColumn(
    "status",
    F.when(F.col("status") == "delayed", 1).otherwise(0)
)
```

Выполняем задание

✓ ПОДБОР ПРИЗНАКОВ И ПРЕОБРАЗОВАНИЕ КАТЕГОРИЙ

```
✓ 3 сек. ▶ from pyspark.ml.feature import StringIndexer, OneHotEncoder
# Индексация для origin
indexer_origin = StringIndexer(inputCol="origin", outputCol="origin_index")
data = indexer_origin.fit(data).transform(data)
# One-Hot Encoding для origin
encoder_origin = OneHotEncoder(inputCol="origin_index", outputCol="origin_vec")
data = encoder_origin.fit(data).transform(data)
# Аналогично для destination
indexer_dest = StringIndexer(inputCol="destination", outputCol="dest_index")
data = indexer_dest.fit(data).transform(data)
| encoder_dest = OneHotEncoder(inputCol="dest_index", outputCol="dest_vec")
data = encoder_dest.fit(data).transform(data)
```

+ Код + Текст

Обработка пропусков

```
✓ [15] # Проверка пропусков
4 from pyspark.sql.functions import col, sum as spark_sum
сек. data.select([spark_sum(col(c).isNull().cast("int")).alias(c) for c in data.columns]).show()
# Импутация (пример для числовых признаков)
from pyspark.ml.feature import Imputer
imputer = Imputer(
    inputCols=["distance_km", "delivery_time_min"],
    outputCols=["distance_km_imp", "delivery_time_min_imp"]
)
data = imputer.fit(data).transform(data)
```

```
⤵ +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|delivery_id|date|origin|destination|distance_km|delivery_time_min|status|origin_index|origin_vec|dest_index|dest_vec|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|          0|  0|  0|          0|          0|          0|  0|          0|          0|          0|          0|
```

Обучение модели с учетом дисбаланса

```
from pyspark.sql import functions as F
from pyspark.ml.classification import LogisticRegression

# 6. Обучение модели с учетом дисбаланса классов
# -----
# Добавляем веса классов ПЕРЕД разделением на train/test
class_weights = {0: 1.0, 1: 5.0} # Вес для класса 1 (delayed) увеличен в 5 раз

data = data.withColumn(
    "class_weight",
    F.when(F.col("status") == 1, class_weights[1]).otherwise(class_weights[0])
)

# Повторяем стратифицированное разделение после добавления class_weight
stratified_data = data.stat.sampleBy(
    "status",
    fractions={0: 0.8, 1: 0.8},
    seed=42
)
train = stratified_data
test = data.subtract(train)

# Теперь столбец class_weight существует в train
lr = LogisticRegression(
    featuresCol="features",
    labelCol="status",
    weightCol="class_weight", # Убедитесь, что имя столбца совпадает
    maxIter=10
)

model = lr.fit(train)
```

```
[ Стратифицирование и разделение ]
```

```
[18] # Ручная стратификация
stratified_data = data.stat.sampleBy(
    "status",
    fractions={0: 0.8, 1: 0.8}, # 80% для каждого класса
    seed=42
)

# Разделение
train = stratified_data
test = data.subtract(train)
```

```
[ Оценка модели ]
```

```
# Посмотрим на распределение предсказаний
predictions.groupBy("prediction").count().show()

# Выведем несколько примеров с реальными метками
predictions.select("status", "probability", "prediction").filter(F.col("status") == 1).show(10)
```

```
+-----+-----+
|prediction|count|
+-----+-----+
|      0.0|   17|
+-----+-----+

+-----+-----+-----+
|status|probability|prediction|
+-----+-----+-----+
+-----+-----+-----+
```

Визуализация

Выполняем основное задание

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import plotly.express as px

# Загрузка данных
df = pd.read_csv('delivery.csv')

# Скачиваем географические регионы России
!wget -q https://raw.githubusercontent.com/codeforamerica/click_that_hood/main/public/data/russia.geojson -O russia.geojson
regions = gpd.read_file("russia.geojson")

# Проверяем названия столбцов в географических
print("Столбцы в географических:", regions.columns.tolist())

# Создаем справочник "город -> регион" (требуется дополнение)
city_to_region = {
    'Москва': 'Москва',
    'Санкт-Петербург': 'Санкт-Петербург',
    'Екатеринбург': 'Свердловская область',
    'Казань': 'Татарстан',
    'Нижний Новгород': 'Нижегородская область',
    'Воронеж': 'Воронежская область',
    'Самара': 'Самарская область',
    'Саратов': 'Саратовская область'
}

# Агрегируем данные по регионам
def get_regional_load(data):
    origins = data['origin'].map(city_to_region).value_counts()
    destinations = data['destination'].map(city_to_region).value_counts()
    return (origins + destinations).fillna(0).astype(int)

regional_load = get_regional_load(df).reset_index(name='count')

# Используем правильное название столбца с регионами (обычно 'name' или 'NAME_1')
regions = regions.rename(columns={'name': 'NAME_1'}) # Если нужно переименовать

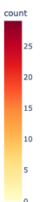
# Объединяем с географическими
merged = regions.merge(
    regional_load,
    left_on='NAME_1',
    right_on='index',
    how='left'
).fillna(0)

# Строим интерактивную карту
fig = px.choropleth(
    merged,
    geojson=merged.geometry,
    locations=merged.index,
    color='count',
    hover_name='NAME_1',
    color_continuous_scale='YlOrRd',
    projection='mercator',
    title='Загруженность доставки по регионам России'
)
fig.update_geos(fitbounds="locations", visible=False)
fig.show()
```

Просим получившуюся карту пояснить за маленьковость

Столбцы в географических: ['name', 'cartodb_id', 'created_at', 'updated_at', 'name_lat', 'geometry']

Загруженность доставки по регионам России




```

# Строим интерактивную карту с увеличенным размером
fig = px.choropleth(
    merged,
    geojson=merged.geometry,
    locations=merged.index,
    color='count',
    hover_name='NAME_1',
    color_continuous_scale='YlOrRd',
    projection='mercator',
    title='Загруженность доставки по регионам России',
    width=1200, # Ширина карты
    height=800  # Высота карты
)

# Дополнительные настройки макета
fig.update_layout(
    margin={"r": 0, "t": 40, "l": 0, "b": 0}, # Убираем лишние отступы
    autosize=False # Отключаем авто-подгонку размера
)

fig.show()

```



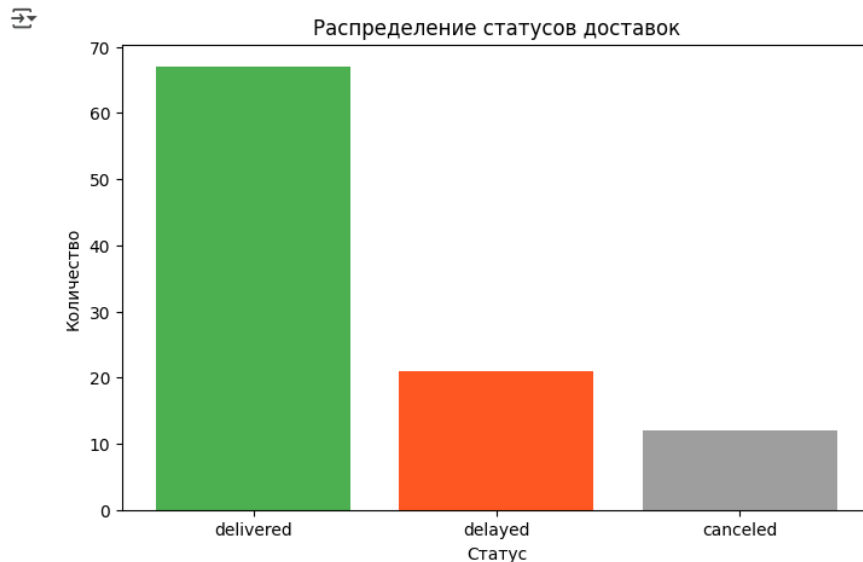
Готовая красота.

А ниже еще визуализации, потому что я их очень люблю

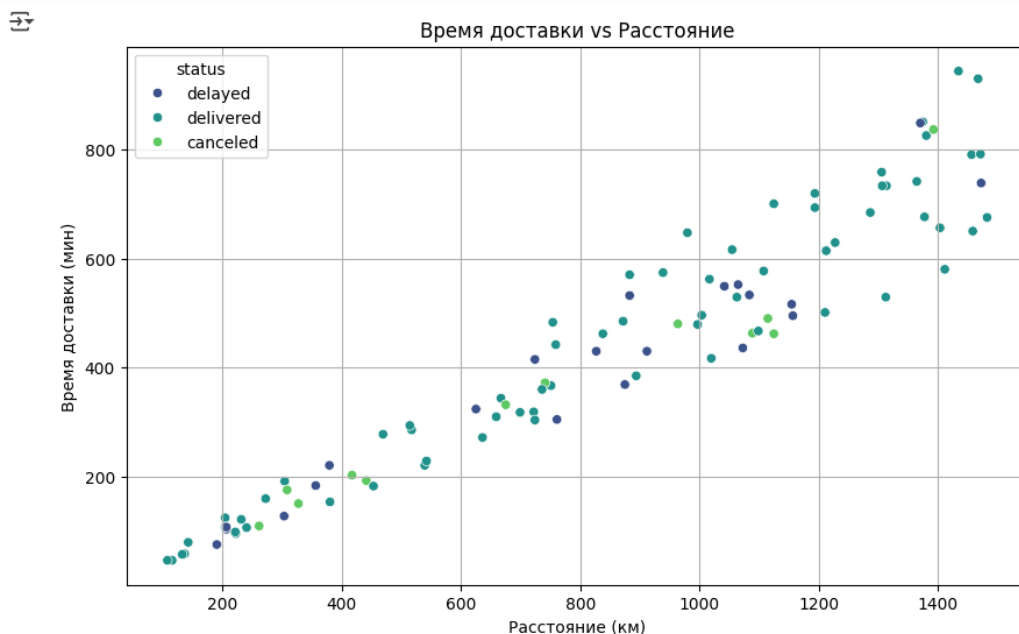
```
[2] import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
```

```
# Чтение данных
df = pd.read_csv('delivery.csv', parse_dates=['date'])
```

```
plt.figure(figsize=(8, 5))
status_counts = df['status'].value_counts()
plt.bar(status_counts.index, status_counts.values, color=['#4CAF50', '#FF5722', '#9E9E9E'])
plt.title('Распределение статусов доставок')
plt.xlabel('Статус')
plt.ylabel('Количество')
plt.show()
```



```
[4] plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='distance_km', y='delivery_time_min', hue='status', palette='viridis')
plt.title('Время доставки vs Расстояние')
plt.xlabel('Расстояние (км)')
plt.ylabel('Время доставки (мин)')
plt.grid(True)
plt.show()
```

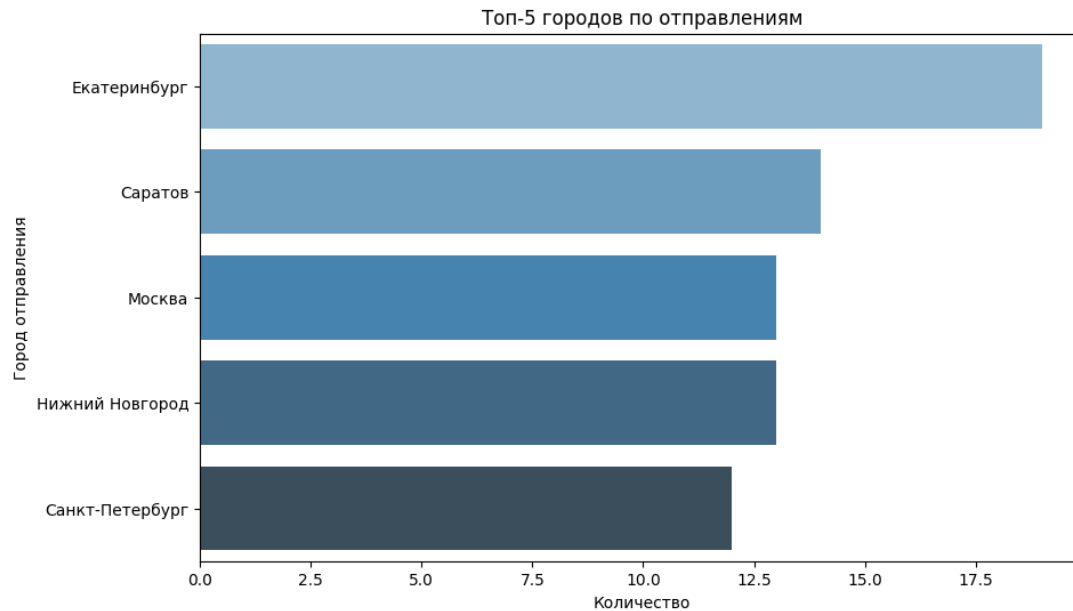


```
plt.figure(figsize=(10, 6))
top_origins = df['origin'].value_counts().head(5)
sns.barplot(x=top_origins.values, y=top_origins.index, palette='Blues_d')
plt.title('Топ-5 городов по отправлениям')
plt.xlabel('Количество')
plt.ylabel('Город отправления')
plt.show()
```

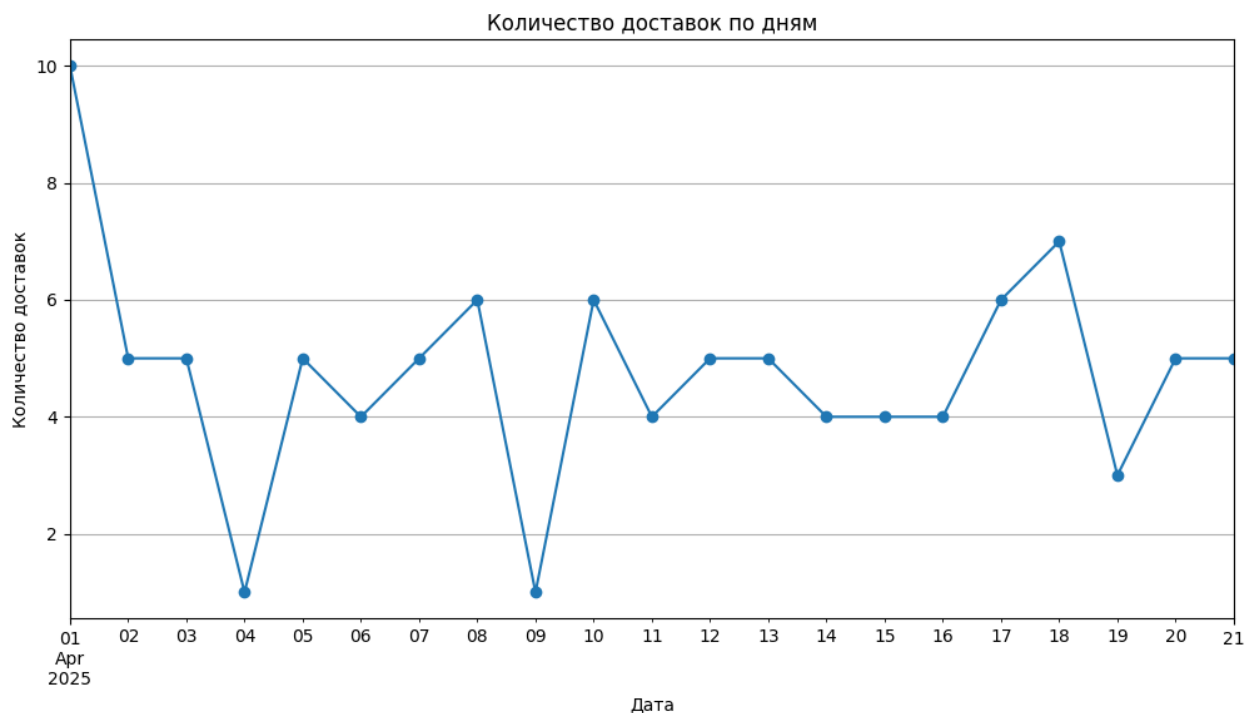
<ipython-input-5-7cd45e4a0608>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` ,

```
sns.barplot(x=top_origins.values, y=top_origins.index, palette='Blues_d')
```



```
plt.figure(figsize=(12, 6))
daily_deliveries = df.groupby('date').size()
daily_deliveries.plot(marker='o', linestyle='--')
plt.title('Количество доставок по дням')
plt.xlabel('Дата')
plt.ylabel('Количество доставок')
plt.grid(True)
plt.show()
```

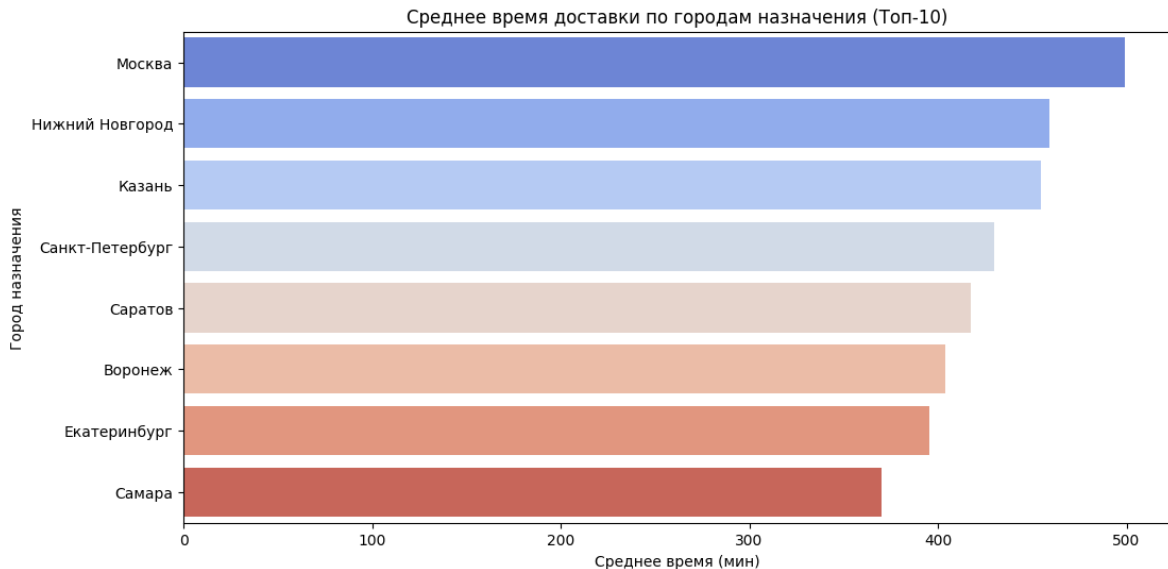


```
plt.figure(figsize=(12, 6))
avg_time = df.groupby('destination')['delivery_time_min'].mean().sort_values(ascending=False).head(10)
sns.barplot(x=avg_time.values, y=avg_time.index, palette='coolwarm')
plt.title('Среднее время доставки по городам назначения (Топ-10)')
plt.xlabel('Среднее время (мин)')
plt.ylabel('Город назначения')
plt.show()
```

<ipython-input-7-9f1d6de6b068>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable

```
sns.barplot(x=avg_time.values, y=avg_time.index, palette='coolwarm')
```

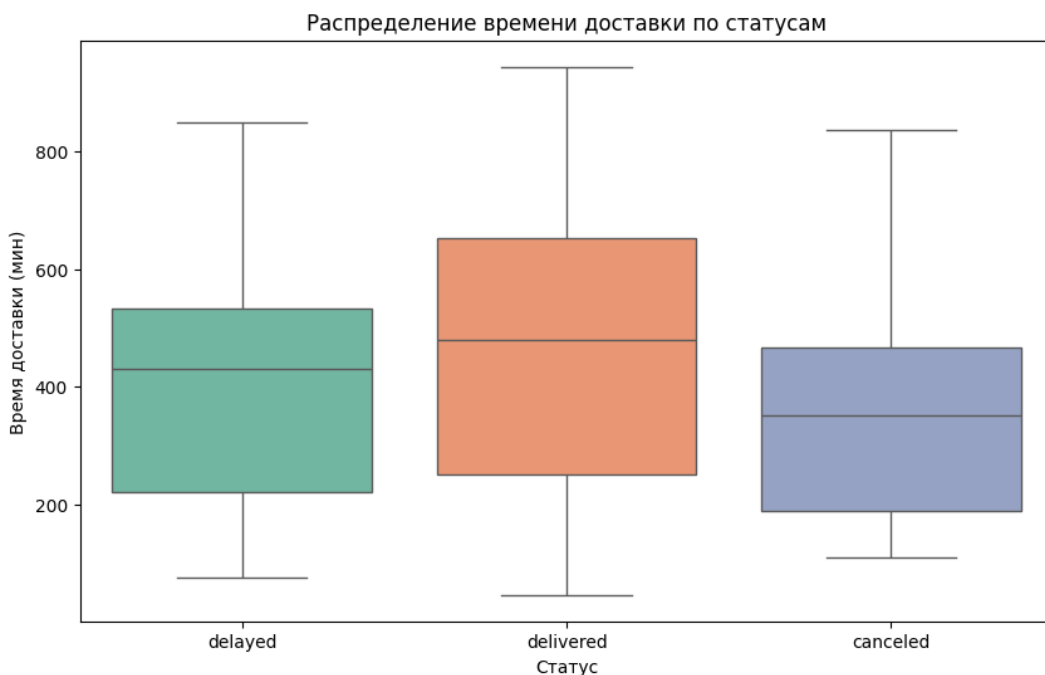


```
8] plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='status', y='delivery_time_min', palette='Set2')
plt.title('Распределение времени доставки по статусам')
plt.xlabel('Статус')
plt.ylabel('Время доставки (мин)')
plt.show()
```

<ipython-input-8-5bfa7d66717c>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

```
sns.boxplot(data=df, x='status', y='delivery_time_min', palette='Set2')
```



Заключение

Вывод:

В ходе выполнения практической работы были успешно освоены базовые принципы работы с Apache Spark и PySpark, что позволило получить ключевые навыки обработки больших данных в распределенной среде. Установка и настройка Spark на локальной машине, а также интеграция с Python через PySpark, продемонстрировали гибкость экосистемы Spark и её совместимость с популярными инструментами анализа данных.

Работа с RDD и DataFrame, включая операции ``map``, ``filter``, ``groupBy`` и ``join``, подтвердила эффективность Spark для параллельной обработки данных.

Использование Spark SQL для выполнения SQL-запросов позволило объединить преимущества реляционных баз данных и распределенных вычислений, упростив агрегацию и анализ сложных наборов данных.

Визуализация результатов с помощью библиотек matplotlib/seaborn и оформление отчета показали, как этапы анализа данных — от предобработки до интерпретации — могут быть объединены в единый рабочий процесс. Важным аспектом стала работа с реальными данными, которая подчеркнула необходимость тщательной очистки и трансформации данных перед анализом.

Практика закрепила понимание преимуществ Spark: масштабируемость, скорость выполнения операций за счет распределенных вычислений и интеграция с экосистемой Python. Полученные навыки позволяют эффективно решать задачи анализа больших данных, что особенно актуально в условиях роста их объемов и сложности. Результаты работы подтверждают, что Spark является мощным инструментом для современных Data Science проектов, а его освоение открывает возможности для работы в области машинного обучения, ETL-процессов и реального времени аналитики.

Материалы практики (код, отчет) доступны в репозитории, что обеспечивает прозрачность и воспроизводимость результатов.