**СОДЕРЖАНИЕ**

[Введение 4](#_Toc199693882)

[1. Теоретическая часть 5](#_Toc199693883)

[1.1 Описание исследуемой области 5](#_Toc199693884)

[1.2 Информация об источнике и о наборе данных 5](#_Toc199693885)

[1.3 Описание процесса сбора данных 8](#_Toc199693886)

[2 Практическая часть 12](#_Toc199693887)

[2.1 Предобработка данных 12](#_Toc199693888)

[2.2 Анализ данных 23](#_Toc199693889)

[2.3 Обработка результатов анализа 26](#_Toc199693890)

[Заключение 30](#_Toc199693891)

[Список используемых источников 31](#_Toc199693892)

[Приложения 32](#_Toc199693893)

[Приложение А 33](#_Toc199693894)

[Приложение Б 34](#_Toc199693895)

[Приложение В 36](#_Toc199693896)

# Введение

**Разработка модели прогнозирования штормов на основе логистической регрессии и анализа временных рядов**

В современном мире точное прогнозирование экстремальных погодных явлений становится критически важной задачей для обеспечения безопасности населения, минимизации экономических потерь и поддержания стабильной работы ключевых отраслей экономики. Среди различных опасных метеорологических явлений штормовые погодные условия занимают особое место, поскольку они характеризуются внезапностью возникновения и способны наносить значительный материальный ущерб, а также создавать угрозу для жизни людей.

Традиционные методы прогнозирования штормов, основанные на физическом моделировании атмосферных процессов, требуют значительных вычислительных ресурсов и не всегда обеспечивают достаточную точность краткосрочных прогнозов. В этой связи применение методов машинного обучения и статистического анализа представляет собой перспективное направление для улучшения качества и заблаговременности штормовых предупреждений.

Особую актуальность приобретает разработка моделей, способных прогнозировать наступление штормовых условий на несколько часов вперед, что позволяет заблаговременно предпринимать необходимые меры безопасности. Среди множества алгоритмов машинного обучения логистическая регрессия демонстрирует высокую эффективность в задачах бинарной классификации, сочетая в себе интерпретируемость результатов и вычислительную эффективность.

Целью данной курсовой работы является разработка программы прогнозирования штормовых явлений на базе алгоритмов логистической регрессии и анализ значимости метеорологических факторов.

Задачи, решаемые в работе:

* Исследовать теоретические основы логистической регрессии и ее применение в задачах классификации
* Проанализировать метеорологические данные и выделить ключевые признаки для прогнозирования штормов
* Разработать программу прогнозирования штормовых условий на различные временные горизонты
* Провести оценку качества модели и анализ значимости метеорологических факторов
* Реализовать визуализацию результатов для практического использования

Практическая значимость работы заключается в создании инструмента, который может быть использован метеорологическими службами для улучшения точности штормовых предупреждений и снижения рисков, связанных с экстремальными погодными явлениями.

# Теоретическая часть

## Прогнозирование штормовых явлений и метеорологические данные

Штормовые явления представляют собой комплекс метеорологических процессов, характеризующихся резким ухудшением погодных условий, включая усиление ветра, выпадение интенсивных осадков, грозовую активность и другие опасные явления. С научной точки зрения, шторм определяется как атмосферное явление, при котором скорость ветра превышает 15 м/с для суши и 20 м/с для морских акваторий.

Ключевые метеорологические параметры, используемые для прогнозирования штормов:

* Скорость ветра - основной индикатор штормовой активности, измеряемый на различных высотах
* Порывы ветра - кратковременные усиления ветра, часто предшествующие развитию шторма
* Атмосферное давление - его резкие изменения свидетельствуют о приближении атмосферных фронтов
* Температура воздуха - вертикальный градиент температуры влияет на конвективную активность
* Влажность воздуха - высокие значения способствуют развитию кучево-дождевых облаков
* Атмосферные фронты - зоны столкновения воздушных масс с различными характеристиками

Источники метеорологических данных включают:

* Станции наземного наблюдения
* Метеорологические радиозонды
* Спутниковые измерения
* Радиолокационные данные
* Результаты численного прогноза погоды

Современные подходы к прогнозированию штормов сочетают физическое моделирование атмосферных процессов с методами статистического анализа и машинного обучения, что позволяет улучшить точность краткосрочных прогнозов и идентифицировать сложные нелинейные зависимости между метеопараметрами.

## Логистическая регрессия в задачах классификации

Логистическая регрессия является одним из наиболее широко применяемых алгоритмов для решения задач бинарной классификации в различных областях, включая метеорологию. Несмотря на название "регрессия", этот метод решает именно задачу классификации, оценивая вероятность принадлежности объекта к определенному классу.

Математические основы логистической регрессии:

Основой алгоритма является логистическая функция (сигмоида), которая имеет вид:

где

— вектор-столбец параметров (весов) логистической регрессии,

— вектор-столбец независимых переменных.

Логистическая функция преобразует линейную комбинацию в вероятность принадлежности к целевому классу, принимающую значения в диапазоне от 0 до 1.

Функция потерь и оптимизация:

Для обучения модели логистической регрессии используется функция log loss (логистические потери):

*,*

где — вероятность принадлежности к целевому классу *i*-ого объекта,

— целевое значение *i*-ого объекта,

n — количество объектов.

Минимизация функции потерь осуществляется с помощью методов оптимизации, таких как градиентный спуск:

где — градиент для функции потерь Log Loss,

— шаг спуска.

* Преимущества логистической регрессии для прогнозирования штормов:
* Интерпретируемость - коэффициенты модели имеют четкую вероятностную интерпретацию
* Вычислительная эффективность - быстрое обучение и прогнозирование
* Устойчивость к шуму - хорошо работает с зашумленными данными
* Калиброванность вероятностей - выходные значения являются истинными вероятностями

Оценка качества модели осуществляется с помощью метрик:

* Accuracy (точность) - общая доля правильных прогнозов
* Precision (точность) - доля истинных штормов среди спрогнозированных
* Recall (полнота) - доль обнаруженных реальных штормов
* F1-score - гармоническое среднее между precision и recall
* ROC-AUC - площадь под кривой ошибок, характеризующая общее качество классификатора

В контексте прогнозирования штормов особенно важны метрики recall и precision, поскольку высокая полнота позволяет минимизировать пропуски опасных явлений, а высокая точность - снизить количество ложных тревог.

# Практическая часть

В данной курсовой работе будет создан Он будет создан на основе схемы на Рисунок 1.1 — Схема конвейера.

## Предобработка данных

Результат представлен на Рисунке 2.1.

Рисунок .1 — Загрузка данных

Результат представлен на Рисунках 2.2-2.3.

Рисунок 2.2 — Переименование столбцов

**Рисунок 2.3 — Переименование столбцов**

. Результат представлен на Рисунке 2.4.

**Рисунок 2.4 — Объединение данных**

Результат представлен на Рисунке 2.5.

**Рисунок 2.5 — Проверка на наличие пропусков и дубликатов**

**Рисунок 2.6 — Создание пустой директории в HDFS**

Результат представлен на Рисунке 2.7.

**Рисунок 2.7 — Импорт и проверка наличия файла**

Результат представлен на Рисунке 2.8.

**Рисунок 2.8 — Создание таблицы и просмотр структуры таблицы в базе данных labs**

Результат представлен на Рисунке 2.9.

**Рисунок 2.9 — Импорт данных в таблицу MariaDB**

Об успешном импорте данных можно увидеть такой вывод на Рисунке 2.10.

**Рисунок 2.10 — Консольные выводы импорта**

Результат select запроса представлен на Рисунке 2.11.

**Рисунок 2.11 — Результат вывода запроса в базе данных**

Процесс загрузки данных представлены на Рисунке 2.12.

**Рисунок 2.12 — Запись данных из MariaDB в Hive**

Результаты представлены на Рисунке 2.13.

**Рисунок 2.13 — Вывод всех имеющихся таблиц в Hive**

Результаты представлены на Рисунке 2.14.

**Рисунок 2.14 — Вывод таблицы в Hive**

Запуск Spark представлена на Рисунке 2.15.

**Рисунок 2.15 — Запуск Spark**

представленного на Рисунке 2.16.

**Рисунок 2.16 — Содержимое файла .bashrc**

Результат представлен на Рисунке 2.17.

**Рисунок 2.17 — Импорт таблицы из Hive в Spark**

Результат представлен на Рисунке 2.18.

**Рисунок 2.18 — Содержимое директории**

Результат представлен на Рисунке 2.19.

**Рисунок 2.19 — Содержимое директории**

Результат представлен на Рисунке 2.20.

**Рисунок 2.20 — Содержимое файла**

Проделанная работа продемонстрирована на Рисунке 2.21.

**Рисунок 2.21 — Создание конфигурационного файла**

Весь файл представлен на Рисунке 2.22.

**Рисунок 2.22 — Конфигурационный файл агента flume**

Команда представлена на Рисунке 2.23.

**Рисунок 2.23 — Создание топика в kafka**

Команда представлена на Рисунке 2.24.

**Рисунок 2.24 — Проверка существующих топиков kafka**

Команда представлена на Рисунке 2.25.

**Рисунок 2.25 — Запуск консольного потребителя kafka**

Результат работы команды представлен на Рисунке 2.26.

**Рисунок 2.26 — Запуск flafka**

Запускаем код python, который работает с flume и тестируем часть конвейера. Результат работы представлен на Рисунке 2.27. Код представлен в Приложение Б.

**Рисунок 2.27 — Отображение данных в консольном потребителе kafka**

## Анализ данных

Результат представлен на Рисунке 2.28.

**Рисунок 2.28 — Общее число смертей**

Код представлен в Приложение В.

**Рисунок 2.29 — Количество смертей, вызванных в результате загрязнения воздуха**

Результат представлен на Рисунке 2.30.

**Рисунок 2.30 — Число смертей от домашних загрязнений**

Код представлен в Приложение В.

**Рисунок 2.31 — Количество смертей, вызванных от домашних загрязнений**

Код представлен в Приложение В.

**Рисунок 2.32 — Динамика смертности**

Результат представлен на Рисунке 2.33.

**Рисунок 2.33 — Страны с самым высоким уровнем смертности**

Код представлен в Приложение В.

**Рисунок 2.34 — Визуализация по максимальному уровню смертности**

## Обработка результатов анализа

Код представлен в Приложение В.

**Рисунок 2.35 — Тепловая карта корреляций**

Код представлен в Приложение В.

**Рисунок 2.36 — Визуализация распределение уровня смертности**

Код представлен в Приложение В.

**Рисунок 2.37 — Получение прогнозов**

Результаты визуализации представлены на Рисунке 2.38.

**Рисунок 2.38 — Визуализация**

Код представлен в Приложение В.

**Рисунок 2.39 — Метрики модели**

# Заключение

Таким образом, в курсовой работе разработан конвейер для предобработки, анализа и визуализации данных с помощью VirtualBox. Использованы продукты экосистемы Apache (Spark, Hadoop), которые обеспечили эффективную обработку больших массивов данных. Разработанный конвейер продемонстрировал свою масштабируемость и гибкость при работе с разнородными данными о загрязнении воздуха.

Была успешно реализована комплексная система анализа данных о загрязнении воздуха и его влиянии на здоровье населения.

В прогнозе видно, что в Индии процент смертности будет уменьшаться.

Цель работы — разработка конвейера данных на основе технологий Big Data для анализа взаимосвязи между загрязнением воздуха и уровнем заболеваемости — была достигнута. В качестве среды для развертывания инструментов используется VirtualBox.

Задачи, решенные в данной курсовой работе:

* Собрать данные из открытых источников (Kaggle).
* Составить конвейер для сбора и передачи данных.
* Выдвинуть гипотезы и проверить их;
* Визуализировать полученные результаты для выявления взаимосвязей между загрязнением воздуха и уровнем заболеваемости.

# Список используемых источников

Теоретическая часть

1. Hadoop: The Definitive Guide" — Tom White (O'Reilly Media, Inc., 2015). [Электронный ресурс]
2. Hive [Электронный ресурс]. Режим доступа: https://hive.apache.org/
3. Apache Sqoop User Guide" — Apache Sqoop PMC (Apache Software Foundation, 2019) [Электронный ресурс]. Режим доступа:
4. Spark [Электронный ресурс]. Режим доступа: https://spark.apache.org/
5. Освоение MariaDB — Федерико Радззоли, Packt Publishing, 2017 [Электронный ресурс].
6. Oracle [Электронный ресурс]. Режим доступа: https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/home-dir.html
7. HDFS [Электронный ресурс] Режим доступа: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
8. DataFrame [Электронный ресурс] Режим доступа: https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html
9. Python [Электронный ресурс] Режим доступа: <https://docs.python.org/3/>
10. Kaggle [Электронный ресурс] Режим доступа: <https://www.kaggle.com/datasets/abmsayem/air-pollution/data>

# Приложения

Приложение А — код на языке программирование Python, в котором происходит предобработка данных.

Приложение Б — код на языке программирование Python, в котором происходит подключение к базе данных, запрос 5-ти % всех строк в таблице с данными каждые 10 секунд и создание файла из полученных строк в формате CSV в папке Spooldir.

Приложение В — код на языке программирование Python, в котором происходит анализ данных.

## Приложение А

# Список значений, которые нужно удалить

entities\_to\_drop = [

    'African Region (WHO)',

    'East Asia & Pacific (WB)',

    'Eastern Mediterranean Region (WHO)',

    'Europe & Central Asia (WB)',

    'European Region (WHO)',

    'G20',

    'Latin America & Caribbean (WB)',

    'Middle East & North Africa (WB)',

    'North America (WB)',

    'OECD Countries',

    'Region of the Americas (WHO)',

    'South Asia (WB)',

    'South-East Asia Region (WHO)',

    'Sub-Saharan Africa (WB)',

    'Western Pacific Region (WHO)',

    'World',

    'World Bank High Income',

    'World Bank Low Income',

    'World Bank Lower Middle Income',

    'World Bank Upper Middle Income'

]

# Фильтруем DataFrame, оставляя только те строки, где Entity НЕ входит в список

merap2 = merap1[~merap1['Entity'].isin(entities\_to\_drop)]

# Добавление столбца id

merap2['id'] = range(1, len(merap2) + 1)

# Перенос столбца id в самое начало

merap2 = merap2[['id'] + [col for col in merap2.columns if col != 'id']]

# Сохранение DataFrame в CSV файл

merap2.to\_csv('air.csv', index=False)

## Приложение Б

import csv

import pymysql

import schedule

import time

import os

# Параметры подключения к базе данных

DB\_HOST = 'localhost'

DB\_PORT = 3306

DB\_USER = 'student'

DB\_PASSWORD = 'student'

DB\_NAME = 'labs'

# Папка для сохранения данных

CSV\_FOLDER = '/home/student/Labs/C3U4/spooldir/'

# Запрос для получения всех строк в таблице

SQL\_QUERY = 'SELECT \* FROM air'

# Функция для выполнения запроса и сохранения данных в CSV-файл

def fetch\_data\_to\_csv():

# Подключение к базе данных

conn = pymysql.connect(host=DB\_HOST, port=DB\_PORT, user=DB\_USER, password=DB\_PASSWORD, database=DB\_NAME)

cursor = conn.cursor()

cursor.execute(SQL\_QUERY)

# Получение данных из базы данных

rows = cursor.fetchall()

# Закрытие соединения

cursor.close()

conn.close()

# Разбиение данных на части по 5%

total\_rows = len(rows)

rows\_limit = round(total\_rows \* 0.05)

for i in range(0, total\_rows, rows\_limit):

# Генерация имени файла

filename = f"data{i//rows\_limit}.csv"

file\_path = os.path.join(CSV\_FOLDER, filename)

# Сохранение данных в CSV-файл

with open(file\_path, 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['id', 'Entity', 'Year', 'Total\_Deaths\_for\_Air\_Pollution', 'Total\_Deaths\_for\_Outdoor\_Air\_Pollution', 'Total\_Deaths\_for\_Household\_Air\_Pollution\_from\_Solid\_Fuels', 'Death\_Rate\_from\_Air\_Pollution\_Per\_100000'])

for row in rows[i:i+rows\_limit]:

writer.writerow(row)

print(f'Data fetched and saved to file {filename}')

# Задача для выполнения запроса и сохранения данных в CSV-файл

def job():

print('Fetching data...')

fetch\_data\_to\_csv()

# Запуск задачи каждые 10 секунд

schedule.every(10).seconds.do(job)

while True:

schedule.run\_pending()

time.sleep(1)

## Приложение В

import os

os.environ['PYSPARK\_SUBMIT\_ARGS'] = '--packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.1.1 pyspark-shell'

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("AppName") \

.config("hive.metastore.uris", "thrift://localhost:9083") \

.enableHiveSupport() \

.getOrCreate()

# Топ-10 стран по общему числу смертей от загрязнения воздуха (2019)

df\_1 = spark.sql("""SELECT Entity, total\_deaths\_for\_air\_pollution

                    FROM hive\_air

                    WHERE Year = 2019

                    ORDER BY total\_deaths\_for\_air\_pollution DESC

                    LIMIT 10;

                 """)

df\_1.show()

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.pyplot import \*

from matplotlib import rcParams

# Сбор данных для графика

data = [row["Entity"] for row in df\_1.collect()]

score = [*float*(row["total\_deaths\_for\_air\_pollution"]) for row in df\_1.collect()]

# Настройка графика

plt.rcParams.update({'font.size': 80})

plt.figure(*figsize*=(100, 80))

plt.bar(data, score)

# Подписи осей и заголовок

plt.xlabel("Страны")

plt.ylabel("Общее число смертей в результате загрязнения воздуха")

plt.title("Общее число погибших в результате загрязнения воздуха за годы")

# Отображение графика

plt.show()

# Топ-10 стран по смертям от домашних загрязнений

df\_5 = spark.sql("""SELECT entity, total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels

                    FROM hive\_air

                    WHERE year=2019

                    ORDER BY total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels DESC

                    LIMIT 10;

                 """)

df\_5.show()

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.pyplot import \*

from matplotlib import rcParams

# Сбор данных для графика

data = [row["entity"] for row in df\_5.collect()]

score = [*float*(row["total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels"]) for row in df\_5.collect()]

# Настройка графика

plt.rcParams.update({'font.size': 90})

plt.figure(*figsize*=(180, 80))

plt.bar(data, score)

# Подписи осей и заголовок

plt.xlabel("Страны")

plt.ylabel("Общее число смертей от домашних загрязнений")

plt.title("Общее число погибших от домашних загрязнений")

# Отображение графика

plt.show()

# Анализ динамики по годам для топ-5 стран

top\_countries = [row["Entity"] for row in df\_1.head(5)]

df\_dynamics = spark.sql(*f*"""

    SELECT Year, Entity, total\_deaths\_for\_air\_pollution

    FROM hive\_air

    WHERE Entity IN ({','.join([*f*"'{c}'" for c in top\_countries])})

    ORDER BY Year

""").toPandas()

plt.figure(*figsize*=(100, 80))

for country in top\_countries:

    country\_data = df\_dynamics[df\_dynamics['Entity'] == country]

    plt.plot(country\_data['Year'], country\_data['total\_deaths\_for\_air\_pollution'], 'o-', *label*=country, *linewidth*=10, *markersize*=50)

plt.title('Динамика смертности от загрязнения воздуха (1990-2019)', *pad*=20)

plt.xlabel('Год')

plt.ylabel('Число смертей')

plt.legend()

plt.grid(True, *linestyle*='--', *alpha*=0.7)

plt.xticks(*rotation*=45)

plt.tight\_layout()

plt.show()

# Топ-10 стран по максимальному уровню смертности

df\_6 = spark.sql("""SELECT entity, death\_rate\_from\_air\_pollution\_per\_100000

                    FROM hive\_air

                    WHERE year=2019

                    ORDER BY death\_rate\_from\_air\_pollution\_per\_100000 DESC

                    LIMIT 10;

                 """)

df\_6.show()

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.pyplot import \*

from matplotlib import rcParams

# Сбор данных для графика

data = [row["entity"] for row in df\_6.collect()]

score = [*float*(row["death\_rate\_from\_air\_pollution\_per\_100000"]) for row in df\_6.collect()]

# Настройка графика

plt.figure(*figsize*=(120, 100))

plt.rcParams.update({'font.size': 60})

plt.bar(data, score)

# Подписи осей и заголовок

plt.xlabel("Страны")

plt.ylabel("Максимальный уровень")

plt.title("Максимальный уровень смертности на 100000")

# Отображение графика

plt.show()

from pyspark.sql.functions import col

import seaborn as sns

import matplotlib.pyplot as plt

# Исходные и целевые названия колонок

numeric\_cols = [

"total\_deaths\_for\_air\_pollution",

"total\_deaths\_for\_outdoor\_air\_pollution",

"total\_deaths\_for\_household\_air\_pollution\_from\_solid\_fuels",

"death\_rate\_from\_air\_pollution\_per\_100000",

"deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent"

]

ru\_names = [

"Общая смертность",

"Смертность от наружного загрязнения",

"Смертность от домашних загрязнений",

"Уровень смертности",

"Доля домашних загрязнений"

]

# Создание DataFrame с нужными типами данных и переименованными колонками

df\_renamed = df.select([col(c).cast('double') for c in numeric\_cols]) \

.toPandas()

df\_renamed.columns = ru\_names # Переименование колонок

# Расчёт корреляционной матрицы

corr\_matrix = df\_renamed.corr()

# Построение тепловой карты

plt.figure(figsize=(120, 100))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', center=0, fmt='.2f')

plt.title('Корреляционная матрица показателей загрязнения воздуха', pad=20)

plt.xticks(rotation=45)

plt.yticks(rotation=0)

plt.tight\_layout()

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

# 1. Распределение уровня смертности

plt.figure(*figsize*=(100, 80))

plt.rcParams.update({'font.size': 80})

sns.histplot(df.toPandas(), *x*='death\_rate\_from\_air\_pollution\_per\_100000', *bins*=30, *kde*=True, *line\_kws*={"linewidth":10})

plt.title('Распределение уровня смертности')

plt.xlabel('Уровень смертности на 100000')

plt.ylabel('Частота')

plt.show()

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.regression import LinearRegression

import pandas as pd

import matplotlib.pyplot as plt

df\_filtered = df.filter(col("entity") == "India")

# Создаем вектор признаков

assembler = VectorAssembler(

*inputCols*=["year"],

*outputCol*="features"

)

df\_train = assembler.transform(df\_filtered)

# Обучаем модель линейной регрессии с регуляризацией

lr = LinearRegression(

*featuresCol*="features",

*labelCol*="deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent",

*regParam*=0.3,        # Параметр регуляризации

*elasticNetParam*=0.8  # Сочетание L1 и L2 регуляризации

)

model = lr.fit(df\_train)

# Создаем датафрейм с будущими годами для прогноза

future\_years = [2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030]

future\_data = spark.createDataFrame(pd.DataFrame({"year": future\_years}))

future\_data = assembler.transform(future\_data)

# Получаем прогнозы

predictions = model.transform(future\_data)

predictions.select("year", "prediction").show()

# Собираем результаты в pandas DataFrame для визуализации

results = predictions.select("year", "prediction").toPandas()

# Подготовка данных для графика

historical\_data = df\_filtered.select("year", "deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent").toPandas()

plt.figure(*figsize*=(100, 40))

plt.rcParams.update({'font.size': 100})

# Исторические данные

plt.scatter(historical\_data["year"],

            historical\_data["deaths\_for\_household\_air\_pollution\_from\_solid\_fuels\_percent"],

*color*='blue', *label*='Исторические данные', *s*=1000)

# Прогнозные значения

plt.plot(results["year"], results["prediction"],

*color*='red', *marker*='o', *linestyle*='--', *markersize*=50,

*label*='Прогноз (линейная регрессия)')

# Настройка графика

plt.title("Прогноз смертности", *fontsize*=80)

plt.xlabel("Год", *fontsize*=80)

plt.ylabel("Процент", *fontsize*=80)

plt.grid(True, *linestyle*='--', *alpha*=0.7)

plt.legend(*fontsize*=80)

plt.tight\_layout()

plt.show()

# Получаем метрики модели

training\_summary = model.summary

print(*f*"R² (коэффициент детерминации): {training\_summary.r2*:.4f*}")

print(*f*"RMSE (среднеквадратичная ошибка): {training\_summary.rootMeanSquaredError*:.2f*}")

print(*f*"Коэффициенты модели:")

print(*f*"  Наклон (slope): {model.coefficients[0]*:.2f*}")

print(*f*"  Пересечение (intercept): {model.intercept*:.2f*}")