Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Кафедра информационных технологий и автоматизированных систем

ОТЧЁТ ПО САМОСТОЯТЕЛЬНОЙ РАБОТЕ № 3

по дисциплине «Технологии анализа данных DataMining»

Тема: «Использование библиотеки numpy»

Выполнил студент гр. АСУ8-24-1м

Пельц Данил Андреевич

(Фамилия И.О.)

(номер зачетной книжки)

Проверил доцент каф. ИТАС

(должность)

Городилов Алексей Юрьевич

(Фамилия И.О)

(оценка)

(дата, подпись)

Пермь 2025 г.

Цель работы: Исследовать и проанализировать данные о производстве, потреблении и зависимости прибыли от скидок с помощью методов обработки массивов и аппроксимации, реализуя вычисления и визуализацию с использованием библиотек numpy, scipy и matplotlib.

Задачи:

1. Загрузить и обработать данные о производстве и потреблении электроэнергии различных стран, а также данные о прибыли в зависимости от скидок.

2. Рассчитать средние показатели производства и потребления электроэнергии за определённый период и определить ключевые статистические характеристики по разным критериям.

3. Построить и решить системы линейных уравнений для нахождения коэффициентов квадратичных и кубических полиномов, аппроксимирующих зависимость прибыли от скидок.

4. Оценить качество полученных моделей с помощью меры квадратичного отклонения (RSS), визуализировать результаты и сделать прогнозы прибыли для заданных значений скидок.

Решение задания 1:

1. Загрузка данных

Для загрузки данных из файлов была применена функция np.genfromtxt. Также были использованы параметры для разделения по столбцам и правильной обработки типов данных (рис. 1):

```
import numpy as np
import matplotlib.pyplot as plt

# читаем названия стран (первый столбец как строки)
strani = np.genfromtxt("global-electricity-generation.csv", delimiter=",", dtype=str, skip_header=1, usecols=0)

# читаем данные генерации (без стран)
dannieGen = np.genfromtxt("global-electricity-generation.csv", delimiter=",", dtype=float, skip_header=1, usecols=range(1,31))

# читаем данные потребления (без стран)
dannieCons = np.genfromtxt("global-electricity-consumption.csv", delimiter=",", dtype=float, skip_header=1, usecols=range(1,31))
```

Рис. 1

2. Среднее ежегодное производство и потребление за последние 5 лет

Было выччеслено среднее значение по последним 5 годам с помощью функции np.nanmean с параметром axis=1 (по строкам) (рис. 2):

```
# последние 5 лет
posled5gen = np.nanmean(dannieGen[:,-5:], axis=1)
posled5cons = np.nanmean(dannieCons[:,-5:], axis=1)
```

Рис. 2

3.1 Суммарное потребление электроэнергии за каждый год

Сумма значений по всем странам для каждого года с помощью np.nansum по оси 0 (рис. 3):

```
# 3.1 суммарное потребление по всем странам за каждый год
vsego_po_godam_cons = np.nansum(dannieCons, axis=0)
```

Рис. 3

3.2 Максимальное количество электроэнергии, произведённое одной страной за один год

Был найден максимум с игнорированием NaN через np.nanmax (рис. 4):

```
# 3.2 макс производство одной страны за один год
maxGen = np.nanmax(dannieGen)
```

Рис. 4

3.3 Страны с производством более 500 млрд кВт*ч в среднем за последние 5 лет (рис. 5)

```
# 3.3 страны >500 млрд
bolshe500 = strani[posled5gen > 500]
```

Рис. 5

3.4 10% стран с наибольшим средним потреблением за последние 5 лет

Был вычислен 90-й процентиль и фильтрация страны по массиву средних потреблений (рис. 6):

```
# 3.4 топ 10% по потреблению
kvantil = np.quantile(posled5cons, 0.9)
top10proc = strani[posled5cons >= kvantil]
```

Рис. 6

3.5 Страны, которые увеличили производство в 2021 году более чем в 10 раз по сравнению с 1992 (рис. 7)

```
# 3.5 2021 > 10 * 1992
god1992 = dannieGen[:,0]
god2021 = dannieGen[:,-1]
tenraz = strani[god2021 > god1992*10]
```

Рис. 7

3.6 Страны, потратившие более 100 млрд кВт*ч и при этом произведшие меньше, чем потратили (рис. 8)

```
# 3.6 тратили >100 и произвели меньше чем тратили
sumGen = np.nansum(dannieGen, axis=1)
sumCons = np.nansum(dannieCons, axis=1)
krutie = strani[(sumCons > 100) & (sumGen < sumCons)]
```

Рис. 8

3.7 Страна с максимальным потреблением в 2020 году (рис. 9)

```
# 3.7 макс потребление в 2020
cons2020 = dannieCons[:,-2]
max2020 = strani[np.argmax(cons2020)]
```

Рис. 9

4. Вывод значений и визуализация суммарного потребления по годам (рис. 10)

```
# выводы
print("3.1 Суммарное потребление за каждый год:", vsego_po_godam_cons)
print("")
print("3.2 Макс производство одной страны за один год:", maxGen)
print("")
print("3.3 >500 млрд производство (среднее за 5 лет):", bolshe500)
print("")
print("3.4 Топ 10% стран по потреблению (среднее за 5 лет):", top10proc)
print("")
print("3.5 В 2021 >10 раз чем в 1992:", tenraz)
print("")
print("3.6 Потребили >100 и произвели меньше:", krutie)
print("")
print("3.7 Больше всех потратил в 2020:", max2020)

# график
plt.plot(vsego_po_godam_cons)
plt.title("Суммарное потребление по годам")
plt.xlabel("Годы (с 1992)")
plt.ylabel("Потребление (млрд кВт*ч)")
plt.show()
```

Рис. 10

Решение задания 2:

1. Загрузка и подготовка данных

Были импортированы данные из файла «data2.csv» с помощью функции np.genfromtxt. Для корректной обработки строк с символами BOM и названиями месяцев реализована очистка и преобразование значений скидок в числовой формат. Это позволило получить массивы скидок и прибыли, пригодные для дальнейшего анализа и апроксимации (рис. 1):

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve

# читаем наш CSV файл, разделитель ;, всё как строки
dd = np.genfromtxt("data2.csv", delimiter=";", dtype=str)

# создаём пустые списки для скидок и прибыли
xik = []
yik = []

# идём по всем строкам из файла
for a, b in dd:
    a = a.replace("п»ї","")  # убираем мусор из первой ячейки (BOM)
    try:
        q = float(a)  # пытаемся превратить скидку в число
    except:
        # если не число, то там типа 01.фев, 01.мар и т.д.
        # делаем словарь месяцев
        mes = {"янв":1,"фев":2,"мар":3,"апр":4,"май":5,"июн":6,
               "июл":7,"авг":8,"сен":9,"окт":10,"ноя":11,"дек":12}
        z = a.split(".")  # делим строку по точке
        try:
            q = float(mes[z[1]])  # берём номер месяца
        except:
            q = 0  # если вообще странно, ставим 0
    xik.append(q)        # добавляем скидку в список
    yik.append(float(b)) # добавляем прибыль в список

# превращаем списки в numpy массивы
xx = np.array(xik)
yy = np.array(yik)
```

Рис. 1

2. Формирование системы уравнений для квадратичного полинома

Для обеспечения точности модели выбрал три точки, равномерно распределённые по диапазону данных (начало, середина, конец). Построена система линейных уравнений, в которой значения полинома в этих точках должны совпадать с исходными данными. Такой выбор обеспечивает более представительное аппроксимирование зависимости (Рис. 2):

```
# выбираем точки для квадратика: первая, середина, последняя
p1 = 0
p2 = len(xx)//2
p3 = len(xx)-1

# строим систему уравнений для квадратичного полинома
A2 = np.array([[xx[p1]**2, xx[p1], 1],
               [xx[p2]**2, xx[p2], 1],
               [xx[p3]**2, xx[p3], 1]])
b2 = np.array([yy[p1], yy[p2], yy[p3]])
```

Рис. 2

3. Решение СЛУ для нахождения коэффициентов квадратичного полинома

Решение системы при помощи scipy.linalg.solve позволило получить коэффициенты a2,a1,a0a2,a1,a0 квадратичного полинома. Это даёт аналитическую формулу, описывающую приближённую зависимость прибыли от скидки (рис. 3):

```
# решаем систему и получаем коэффициенты a, b, c
koef2 = solve(A2,b2)
print("коэф квадр:", koef2)
```

Рис. 3

4. Вычисление значений квадратичного полинома на всех точках и оценка ошибки

Полученный полином применён ко всем исходным значениям скидок, что позволило сравнить прогнозируемые и реальные прибыли. Подсчитано квадратичное отклонение (RSS), что служит мерой качества аппроксимации — чем меньше RSS, тем лучше модель описывает данные (рис. 4):

```
# считаем значения полинома на всех точках
f2 = koef2[0]*xx**2 + koef2[1]*xx + koef2[2]

# считаем RSS (квадратичное отклонение)
rss2 = np.sum((yy - f2)**2)
print("RSS квадр=", rss2)
```

Рис. 4

5. Формирование системы уравнений для кубического полинома

Для улучшения точности модели добавлена четвертая точка, что соответствует полиному третьей степени. Аналогично сформирована и решена система линейных уравнений, что позволяет получить более гибкую формулу зависимости (рис. 5):

```
# теперь кубический полином, выбираем ещё одну точку
p4 = len(xx)//3*2

A3 = np.array([[xx[p1]**3, xx[p1]**2, xx[p1], 1],
               [xx[p2]**3, xx[p2]**2, xx[p2], 1],
               [xx[p3]**3, xx[p3]**2, xx[p3], 1],
               [xx[p4]**3, xx[p4]**2, xx[p4], 1]])
b3 = np.array([yy[p1], yy[p2], yy[p3], yy[p4]])

# решаем систему для кубика
koef3 = solve(A3,b3)
print("коэф куб:", koef3)
```

Рис. 5

6. Вычисление значений кубического полинома и оценка ошибки

Расчёт значений кубического полинома позволил провести сравнение с реальными данными и вычислить RSS для оценки точности модели. Это даёт основание оценить, насколько сложная модель лучше или хуже отражает зависимость (рис. 6):

```
# считаем значения кубического полинома
f3 = koef3[0]*xx**3 + koef3[1]*xx**2 + koef3[2]*xx + koef3[3]

# RSS для кубика
rss3 = np.sum((yy - f3)**2)
print("RSS куб=", rss3)
```

Рис. 6

7. Визуализация исходных данных и обеих аппроксимаций

Построены графики, позволяющие наглядно сравнить исходные данные с моделями. Визуальное представление помогает понять, какая из моделей лучше подходит под данные и выявить особенности аппроксимации (рис. 7):

```
# рисуем графики
plt.scatter(xx,yy,color="red",label="исходные") # точки из файла
plt.plot(xx,f2,label="квадрат")                 # квадратичная линия
plt.plot(xx,f3,label="кубик")                   # кубическая линия
plt.legend()
plt.show()
```

Рис. 7

8. Выбор лучшей модели по минимальному RSS и прогноз прибыли при новых скидках

Сравнение RSS двух моделей позволило выбрать наиболее точную. На её основе рассчитаны прогнозируемые значения прибыли при скидках 6% и 8%. Это демонстрирует практическое применение модели для бизнес-прогнозов (рис. 8):

```
# выбираем лучший вариант (меньше RSS)
if rss2<rss3:
    print("лучше квадратичный")
    pred6 = koef2[0]*6**2 + koef2[1]*6 + koef2[2]  # считаем прибыль при скидке 6
    pred8 = koef2[0]*8**2 + koef2[1]*8 + koef2[2]  # при 8%
else:
    print("лучше кубический")
    pred6 = koef3[0]*6**3 + koef3[1]*6**2 + koef3[2]*6 + koef3[3]
    pred8 = koef3[0]*8**3 + koef3[1]*8**2 + koef3[2]*8 + koef3[3]

# выводим прогнозы
print("при 6% =", pred6)
print("при 8% =", pred8)
```

Рис. 8

Вывод:

В лабораторной работе я разобрался, как загрузить и обработать данные по электроэнергии и прибыли из разных файлов. Считал средние значения за последние годы и нашёл интересные показатели, вроде стран с большим производством или высоким потреблением. Затем построил простые модели — квадратичную и кубическую — чтобы понять, как прибыль зависит от скидок. Для этого решил систему уравнений, чтобы получить формулы, которые приблизительно описывают данные.

Потом проверил, какая модель лучше, считая ошибку (RSS), и сравнил графики. Оказалось, что одна из моделей подходит точнее, и на её основе сделал прогнозы прибыли при новых скидках. В итоге эта работа показала, что даже простые математические приёмы с помощью кода помогают получить полезные выводы из цифр и понять, как меняются данные.